Les enregistrements et les fichiers

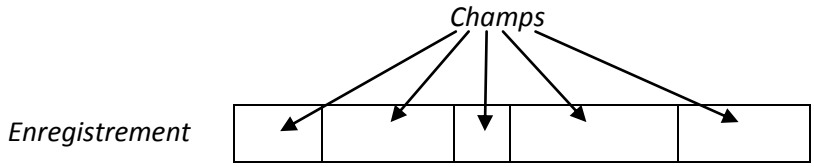
1. Structures de données hétérogènes

Une *structure de données* ou bien un *type structuré* correspond à des variables *composées d'un nombre de données élémentaires* reliées d'une certaine façon. Quand les éléments (données élémentaires) sont de types différents, la structure de données est dite *hétérogène.*

2. Les enregistrements

2.1. Définition

Contrairement aux tableaux, les enregistrements sont des structures de données hétérogènes. Les éléments qui constituent un enregistrement sont appelés *champs*. Chaque champ est identifié par un nom.



2.2. Déclaration

Pour déclarer un enregistrement:

- On déclare d'abord un type structuré,
- ensuite on fait la déclaration de l'enregistrement à partir du ce type structuré.

Un enregistrement est une variable d'un type structuré.

Syntaxe:

```
Type <nom_type> = Enregistrement
        <champ1> : <type1>
        <champ2> : <type2>
            …
        <champN> : <typeN>
     Fin
Var  <nom_enreg> : <nom_type>
```

Déclaration du type structuré

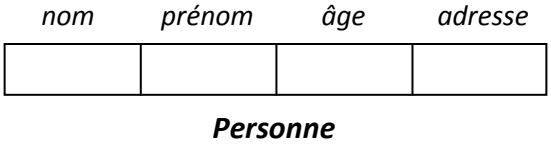
Déclaration de l'enregistrement

Avec:

- ✓ `<nom_type>` est le nom du type structuré;
- ✓ `<champ1>`, …, `<champN>` sont les noms des champs de l'enregistrement;
- ✓ `<type1>`, …, `<typeN>` sont les types des champs de l'enregistrement;
- ✓ `<nom_enreg>` est le nom de l'enregistrement.

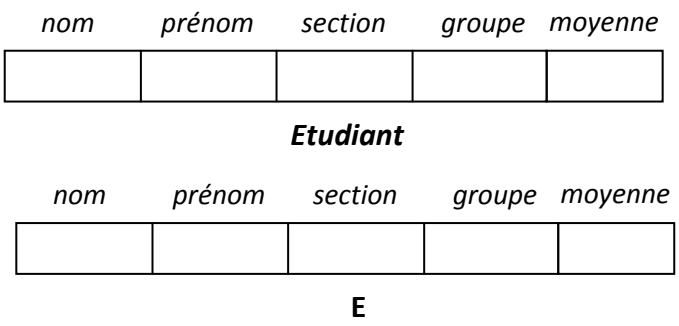
Exemple 1: déclaration d'un type structuré *Personne* et d'un enregistrement *P* de type *Personne*.

```
Type Personne = Enregistrement
       nom, prénom : chaîne
       âge : entier
       adresse : chaîne
     Fin
Var  P : Personne
```



Exemple 2: déclaration d'un type structuré *Etudiant* et d'un enregistrement *E* de type *Etudiant*.

```
Type Etudiant = Enregistrement
       nom, prénom : chaîne
       section : car
       groupe : chaîne
       moyenne : réel
     Fin
Var  E : Etudiant
```



2.3. Accès aux champs d'un enregistrement

Les champs d'un enregistrement sont accessibles à travers leurs noms, grâce à l'opérateur «**.**» de la manière suivante:

```
<nom_enregistrement>.<nom_champ>
```

Exemple: pour accéder au champ `âge` de la personne **`P`** on écrit:

```
P.âge
```

2.4. Opérations sur les enregistrements

Il existe des opérations sur les enregistrements prises globalement (opérations globales) et des opérations sur les champs de l'enregistrement.

a. Opérations globales

La seule opération globale (sur la totalité de l'enregistrement) possible est l'affectation de deux enregistrements, à condition que les deux enregistrements soient de **même type**.

Exemple: si `P1` et `P2` sont deux enregistrements de type `Personne`, on pourra faire les opérations d'affectation :

`P1 ← P2` ou bien

`P2 ← P1`

Remarques:

- On ne peut pas lire ou écrire globalement un enregistrement. Pour ce faire, il faudra lire champ par champ.

Exemple: pour lire l'enregistrement `P` de type `Personne` on écrit :

`Lire(P.nom,P.prénom,P.âge,P.adresse)` et **NON PAS** `Lire (P).`

- On ne peut pas tester l'égalité (=) et la différence (≠) de deux enregistrements de même type. La comparaison doit être faite champ par champ.

b. Opérations sur champs

Les opérations possibles sur un champ sont les mêmes qui sont réalisables sur une variable qui a le même type que ce champ.

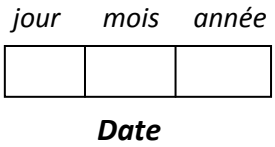
Exemple: soit l'enregistrement `P` de type `Personne`. Le champ `P.âge` est un champ de type entier. Donc, on peut faire les opérations de *lecture*, *écriture*, *affectation*, *comparaison*, *soustraction*, … *etc.* sur ce champ:

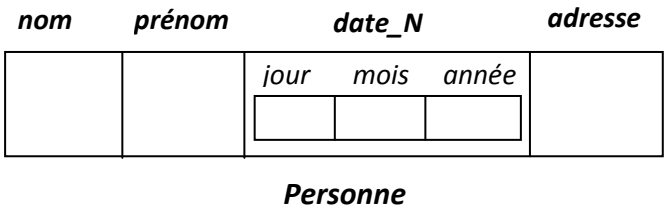
- `Lire (P.âge)`
- `Ecrire (P.âge)`
- `Si (P.âge >18) Alors …`
- `P.âge ← année_actuelle - année_naissance`
- … etc.

2.5. Un enregistrement comme champ d'un autre enregistrement

Un champ d'un enregistrement peut être à son tour un enregistrement. Supposons que dans la structure `Personne`, nous voulions remplacer l'âge de la personne par sa date de naissance. Une date est composée de trois variables (*jour*, *mois*, *année*). Donc, la date peut être représentée par une structure de données.

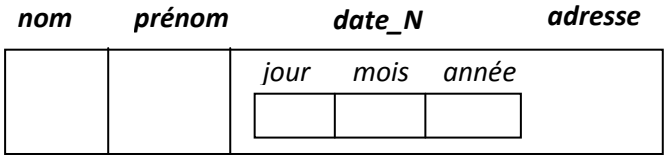
```
Type Date = Enregistrement
       | jour : entier
       | mois : entier
       | année : entier
     Fin

     Personne = Enregistrement
       | nom, prénom : chaîne
       | date_N : Date
       | adresse : chaîne
     Fin

Var  P : Personne
```





❖ Pour accéder aux champs `jour`, `mois` et `année` de l'enregistrement **`P`** on écrit:

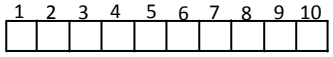
```
P.date_N.jour
P.date_N.mois
P.date_N.année
```



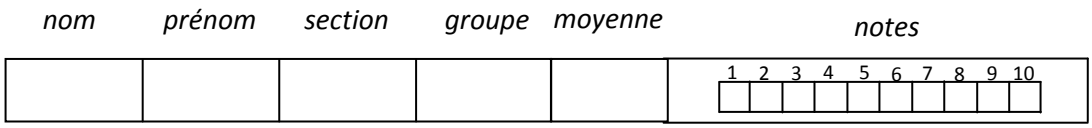
2.6. Un tableau comme champ d'un enregistrement

Un champ d'un enregistrement peut être un tableau. Supposons que dans le type `Etudiant`, nous voulions ajouter les notes de l'étudiant en 10 modules. Les notes de l'étudiant seront stockées dans un tableau.

```
Type TabNotes = tableau [1..10] de réel

     Etudiant = Enregistrement
       | nom, prénom : chaîne
       | section : car
       | groupe : chaîne
       | moyenne : réel
       | notes : TabNotes
     Fin

Var  E : Etudiant
```



TabNotes



Etudiant

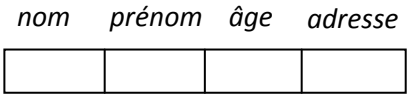
- Pour accéder à la $3^{ème}$ note de l'étudiant **E** on écrit:
  `E.notes[3]`
- Pour accéder à la $7^{ème}$ note de l'étudiant **E** on écrit:
  `E.notes[7]`

2.7. Un tableau d'enregistrements

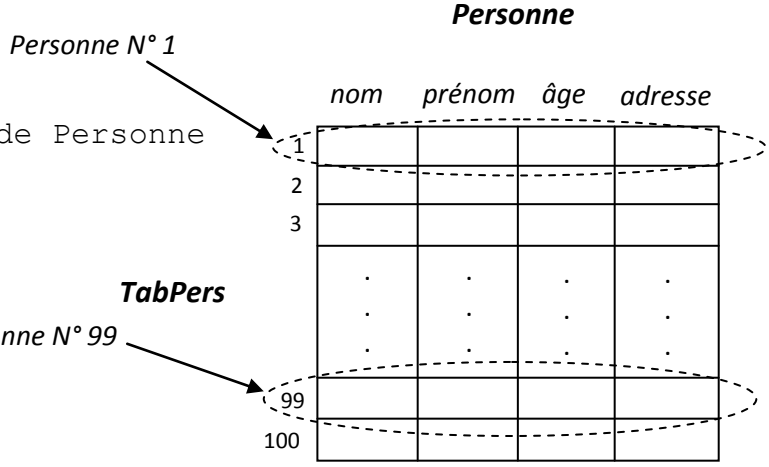
Un groupe d'enregistrements de même type peut être stocké dans un tableau. Dans ce cas, les éléments du tableau seront des enregistrements.

Exemple: supposons que nous voulions stocker les informations sur 100 personnes dans un tableau. Les structures de données nécessaires seront définies comme suit:

```
Type Personne = Enregistrement
       | nom, prénom : chaîne
       | âge : entier
       | adresse : chaîne
     Fin

     TabPers = tableau [1..100]de Personne

Var  T : TabPers
```



Personne



TabPers

❖ Pour accéder à l'adresse de la $2^{ème}$ personne du tableau **T** on écrit:
`T[2].adresse`

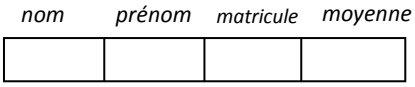
❖ Pour accéder à l'adresse de la $i^{ème}$ personne du tableau **T** on écrit:
`T[i].adresse`

Exercice

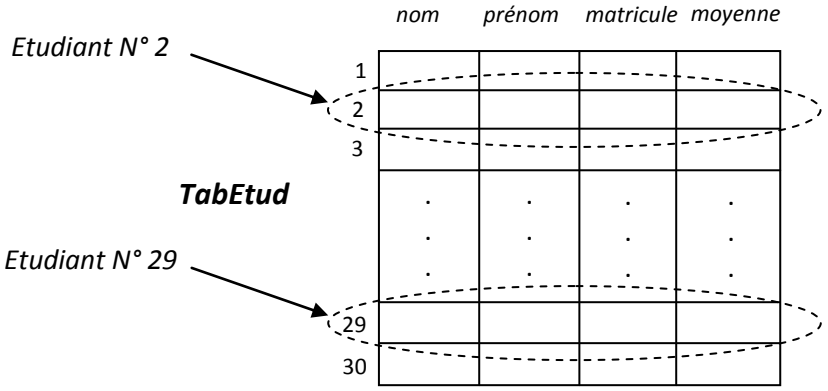
Un étudiant est caractérisé par son *nom*, son *prénom*, son *matricule* sa *moyenne*. Les informations relatives aux étudiants d'un groupe sont stockées dans un tableau de 30 éléments.

1. Définir les types structurés nécessaires;
2. Faire la déclaration des variables nécessaires;
3. Saisir (lire) les informations correspondantes à tous les étudiants de ce groupe;
4. Afficher les noms et les moyennes des étudiants ayants une moyenne ≥ 10.

Solution



Etudiant



TabEtud

1. Les types structurés nécessaires sont:

`Etudiant` : type qui représente les étudiants.

`TabEtud` : type qui représente les tableaux permettant de stocker les informations sur 30 étudiants.

```
Algorithme étudiants
Type Etudiant = Enregistrement
       | nom, prénom : chaîne
       | matricule : entier
       | moyenne : réel
       Fin
     TabEtud = tableau [1..30]de Etudiant
Var T:TabEtud
    i:entier                                              } 2
Début
  Pour i=1 à 30 Faire
    | Lire (T[i].nom,T[i].prénom,T[i].matricule,T[i].moyenne)   } 3
  FinPour
  Pour i=1 à 30 Faire
    | Si(T[i].moyenne ≥ 10)Alors
    |    | Ecrire(T[i].nom)
    |    | Ecrire(T[i].moyenne)
    | FinSi                                                 } 4
  FinPour
Fin
```

Structure de données: Etudiant

Structure de données: TabEtud

3. Les fichiers

3.1. Notion de fichier

- ✓ Un fichier (*file*) est *une collection de données* de même type stocké en général sur un support externe (disquette, disque dur, disque optique, flash disque, ...etc.).
- ✓ Les fichiers permettent de stocker des informations de manière permanente sur le disque, ces informations restent même lorsque l'ordinateur est éteint.

3.2. Les modes d'accès aux fichiers

Le mode d'accès au fichier est la manière d'accéder aux données de ce fichier. Ils existent trois modes d'accès, qui sont:

- ❖ ***L'accès séquentiel*:** pour accéder à une information particulière, il faut lire (parcourir) toutes les informations situées avant cette information.
- ❖ ***L'accès direct (aléatoire)*:** on peut accéder directement à l'information désirée, grâce à son numéro d'ordre.
- ❖ ***L'accès indexé*:** on ne recherche pas directement l'information dans le fichier. On parcourt *un index* pour rechercher une *clef*, et on obtient ainsi l'adresse exacte de l'information recherchée.

3.3. Les fichiers séquentiels

a. Déclaration d'un fichier

En algorithmique:	*En Pascal*:
Var <nom_fichier> : **fichier de** <type_éléments>	**Var** <nom_fichier> : **file of** <type_éléments>;

- ✓ `<nom_fichier>`: est le *nom logique* du fichier.
- ✓ `<type_éléments>`: est le type des éléments qui constituent le fichier.

Exemples:

- ➢ Déclaration d'un fichier qui contient des entiers:

En algorithmique:	*En Pascal*:
Var f : **fichier de** entier	**Var** f : **file of** integer;

- Déclaration d'un fichier qui contient des personnes:

En algorithmique:

```
type Personne = enregistrement
        nom, prénom : chaîne
        âge : entier
        adresse : chaîne
      fin
var f : fichier de Personne
```

En Pascal:

```
type Personne = record
        nom, prénom : string;
        age : integer;
        adresse : string;
      end;
var f : file of Personne;
```

b. Nom logique et nom physique

- *Nom logique* : c'est le nom de fichier utilisé dans le programme. **Exemple:** f.
- *Nom physique* : c'est le nom de fichier stocké sur le disque. **Exemple:** "D:\TP\MonFichier.dat".

c. Opération d'association (assignation)

C'est une opération qui consiste à associer le nom logique au nom physique du fichier.

En algorithmique:

```
Associer (nom_logique, nom_physique)
```

Exemple:

```
Associer (f , "D:\TP\MonFichier.dat")
```

En Pascal:

```
Assign (nom_logique, nom_physique);
```

Exemple:

```
Assign (f , 'D:\TP\MonFichier.dat');
```

d. Pointeur de fichier

Le *pointeur de fichier* est un repère (un marqueur) qui sert à indiquer l'adresse à laquelle il faudra lire ou écrire le prochain enregistrement (élément).

e. Ouverture d'un fichier

- ✓ **L'instruction *Recréer*:** cette instruction permet d'ouvrir un fichier *en écriture*, c'est-à-dire de *créer* le fichier, et d'autoriser des opérations d'*écriture* dans ce dernier. Si le fichier existe déjà, son contenu sera effacé.

En algorithmique:

```
Recréer (nom_logique)
```

En Pascal:

```
Rewrite (nom_logique);
```

- ✓ **L'instruction *Ouvrir*:** cette instruction permet de renvoyer le pointeur au début du fichier pour pouvoir lire ou écrire à partir du début du fichier.

En algorithmique:

```
Ouvrir (nom_logique)
```

En Pascal:

```
Reset (nom_logique);
```

f. Ecriture dans un fichier

L'instruction « **Ecrire** » permet d'écrire ou de modifier une valeur ou un enregistrement dans un fichier.

En algorithmique:

Ecrire (nom_logique, variable)

Exemples:

Ecrire (f , x)

En Pascal:

Write (nom_logique,variable);

Exemples:

Write (f , x);

g. Lecture à partir d'un fichier

L'instruction « **Lire** » permet de lire une valeur ou un enregistrement à partir d'un fichier et de le (la) mettre dans une variable.

En algorithmique:

Lire (nom_logique, variable)

Exemple:

Lire (f , x)

En Pascal:

Read (nom_logique,variable);

Exemple:

Read (f , x);

h. Fermeture d'un fichier

L'instruction « **Fermer** » permet de fermer le fichier ouvert.

En algorithmique:

Fermer (nom_logique)

En Pascal:

Close (nom_logique);

i. Test de fin de fichier

L'instruction « **FdF** » permet de savoir si on a atteint la fin du fichier ou non.

En algorithmique:

FdF (nom_logique)

En Pascal:

EoF (nom_logique)

4. **FdF** (nom_logique) = vrai : on a atteint la fin du fichier.
5. **FdF** (nom_logique) = faux : on n'a pas atteint la fin du fichier.

Les fichiers texte

- ✓ Les fichiers texte sont des *cas particuliers* de fichiers;
- ✓ Les éléments qui forment un fichier texte sont *les caractères*;
- ✓ Les caractères contenus dans un fichier texte sont organisés en lignes;
- ✓ Chaque ligne termine par une *marque de fin de ligne*.
- ✓ Le fichier se termine par une *marque de fin de fichier* (après la dernière ligne).

Déclaration d'un fichier texte

En algorithmique:

```
Var <nom_fichier> : texte
```

En Pascal:

```
Var <nom_fichier> : text;
```

Exemple de lecture et d'affichage d'un fichier texte

En algorithmique:

```
algorithme lecture_texte
var f : texte
    s: chaîne
début
 associer(f, "D:\MonFichier.txt")
 ouvrir(f)
 TantQue non FdF(f) faire
  lire(f,s)
  écrire(s)
 FinTQ
 fermer(f)
fin
```

En Pascal:

```
program lecture_texte;
uses crt;
var f : text;
    s: string;
begin
 clrscr;
 assign(f, 'D:\MonFichier.txt');
 reset(f);
 while not EoF(f) do
  begin
   readln(f,s);
   writeln(s);
  end;
 close(f);
 readkey;
End.
```