

Algorithmique et Structures de Données

L2 MPI-MI

Dr Ousmane DIALLO



Chap8: Les structures d'enregistrement

8.1. Notion d'enregistrement

La structure de **tableau** permet de traiter un nombre **fini** d'informations **de même type** auxquelles on peut accéder par un **indice**. Or souvent il est nécessaire d'organiser de façon hiérarchique un ensemble de données de **types différents mais qualifiant un même objet** comme par exemple :

- les dates chronologiques (année, mois, jour),
- les fiches bibliographiques (titre du livre, auteur, date de parution),
- les fiches personnelles (nom, prénom, âge, sexe, taille).

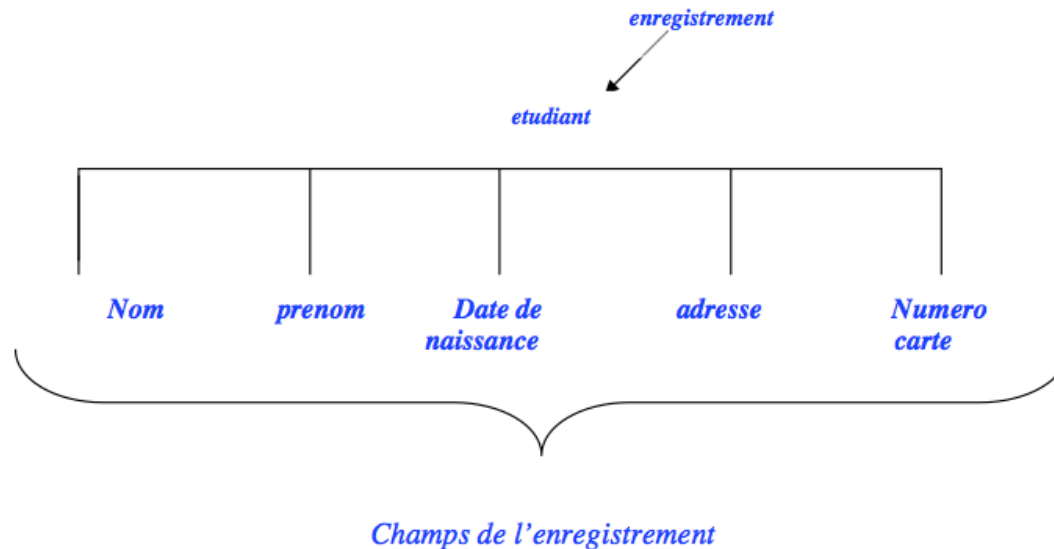
La nature différente de ces éléments (données) conduit le programmeur à utiliser une structure permettant la **définition explicite de chacun de ces éléments: structure enregistrement**.

8.1.1. Définition

Un **enregistrement** est un **type structuré** formé d'un ou de plusieurs éléments pouvant être **de types différents**. Chaque enregistrement est désigné par un **identificateur** (son nom). Chacun des éléments de l'enregistrement occupe un emplacement appelé **champ** désigné à son tour par un identificateur et caractérisé par un **type**. Une **information** représente la valeur du champ.

Chap8: Les structures d'enregistrement

Exemple :



etudiant est la **variable enregistrement**, Nom, prenom, date de naissance, adresse, numero carte sont les **champs de l'enregistrement**.

8.2. Déclaration d'enregistrement

8.2.1. Définition d'un type d'enregistrement

On procède à la définition d'un **type enregistrement** puis à la déclaration d'une **variable enregistrement** de ce type selon la syntaxe suivante :

Chap8: Les structures d'enregistrement

En pseudo-code:

TYPE id_enr = **ENREGISTREMENT**

Chp1 : type1;

Chp2 : type2;

...

Chpn : typen ;

FIN

Variable idv : id_enr ;

Où :

- **id_enr** est l'identificateur de type enregistrement,
- **chp1, chp2, ..., chpn** les champs de l'enregistrement,
- **type1, type2, type3, ..., typen** les types respectifs des champs **chpi** (**i = 1,...,n**),
- **idv** est l'identificateur de la variable enregistrement.

En Pascal:

TYPE id_enr = **RECORD**

Chp1 : type1;

Chp2 : type2;

...

Chpn : typen ;

END ;

Var idv : id_enr ;

Chap8: Les structures d'enregistrement

Exemple:

```
TYPE etud = RECORD
```

```
    nom, prenom : string[30];
```

```
    date_naissance : string[8];
```

```
    adresse       : string[50];
```

```
    numero_carte  : integer;
```

```
END;
```

```
VAR etudiant : etud ;
```

Nous avons défini un type enregistrement **etud** et déclaré une variable enregistrement **etudiant** de type **etud**.

Remarque: La définition d'un type enregistrement ne crée pas une variable enregistrement. Pour créer une variable enregistrement, il faut procéder à sa déclaration dans la section **VAR**.

Exercice d'application:

Définir un type d'enregistrement pour représenter une voiture. On suppose qu'une voiture est caractérisée par sa marque, sa date de fabrication, son numéro d'immatriculation et le nombre de places.

Chap8: Les structures d'enregistrement

TYPE voiture = RECORD

marque : string[25];

date_fabrication : string[8];

immatriculation : string[8];

nbre_places : integer;

END;

8.2.2. Déclaration d'une variable enregistrement sans définition préalable d'un type d'enregistrement

Syntaxe:

VAR idv : RECORD

Chp1 : type1;

Chp2 : type2;

...

Chpn : typen;

END ;

Exemple:

VAR voiture : RECORD

marque : string[25];

date_fabrication : string[8];

immatriculation : string[8];

nbre_places : integer;

END ;

Cette déclaration crée une **variable enregistrement** voiture composée de quatre champs.

Chap8: Les structures d'enregistrement

8.3. Accès aux champs d'un enregistrement

L'accès (ou référence) aux champs d'un enregistrement se fait de deux façons : par **nom composé** ou par l'instruction **WITH**.

8.3.1. Accès par nom composé

L'accès à un champ par cette méthode se fait donc en nommant la **variable enregistrement** puis le **champ** et séparant les deux noms par un point «. ».

Syntaxe:

`idev.chpi ;`

où **idev** est l'identificateur de la variable enregistrement et **chpi** est l'identificateur du champ.

Exemple:

Voici les instructions permettant de remplir les champs de l'enregistrement étudiant déclaré précédemment (section 8.2.1) :

Chap8: Les structures d'enregistrement

```
WRITE(' Donner le nom :' ) ;  
READLN(etudiant.nom);  
WRITE('Donner le prenom :' ) ;  
READLN(etudiant.prenom);  
WRITE('Donner la date de naissance:' ) ;  
READLN(etudiant.date_naissance);  
WRITE('Donner l"adresse :' ) ;  
READLN(etudiant.adresse);
```

Exercice d'application

Écrire les instructions permettant d'afficher les champs de l'enregistrement voiture déclaré précédemment (section 8.2.2).

8.3.2 Accès par l'instruction WITH

Le langage pascal permet, avec l'utilisation de l'instruction **WITH**, d'éviter la répétition de l'identificateur de la variable enregistrement.

Syntaxe:

WITH idv **DO** **<instructions>** où idv est l'identificateur de la variable enregistrement **<instructions>** est une instruction simple ou composée (dans ce cas il faut la placer entre un **BEGIN** et **END**).

Chap8: Les structures d'enregistrement

Exemple:

Reprenons l'exemple précédent de la section 8.3.1

WITH etudiant **DO**

BEGIN

WRITE(' Donner le nom : ') ;

READLN(nom);

WRITE('Donner le prenom : ') ;

READLN(prenom);

WRITE('Donner la date de naissance:') ;

READLN(date_naissance);

WRITE('Donner l"adresse : ') ;

READLN(adresse);

END;

Exercice d'application:

Reprendre l'exercice de la section précédente (8.3.1).

Chap8: Les structures d'enregistrement

8.4. Enregistrement d'enregistrements

- On parle d'enregistrement d'enregistrements lorsqu'un champ est lui-même composé d'autres champs, autrement dit, le **champ est de type enregistrement**.
- Par exemple, définissons une structure qui permet d'enregistrer les informations relatives à un étudiant en considérant la **date de naissance** et l'**adresse** comme des enregistrements.

```
TYPE date = RECORD
```

```
    Jour   : integer ;
```

```
    Mois   : string[12] ;
```

```
    Annee  : integer ;
```

```
END ;
```

```
adr = RECORD
```

```
    numero_rue: integer ;
```

```
    rue       : string[50] ;
```

```
    ville     : string[30];
```

```
END ;
```

```
etudiant = RECORD
```

```
    Nom, prenom   : string[30] ;
```

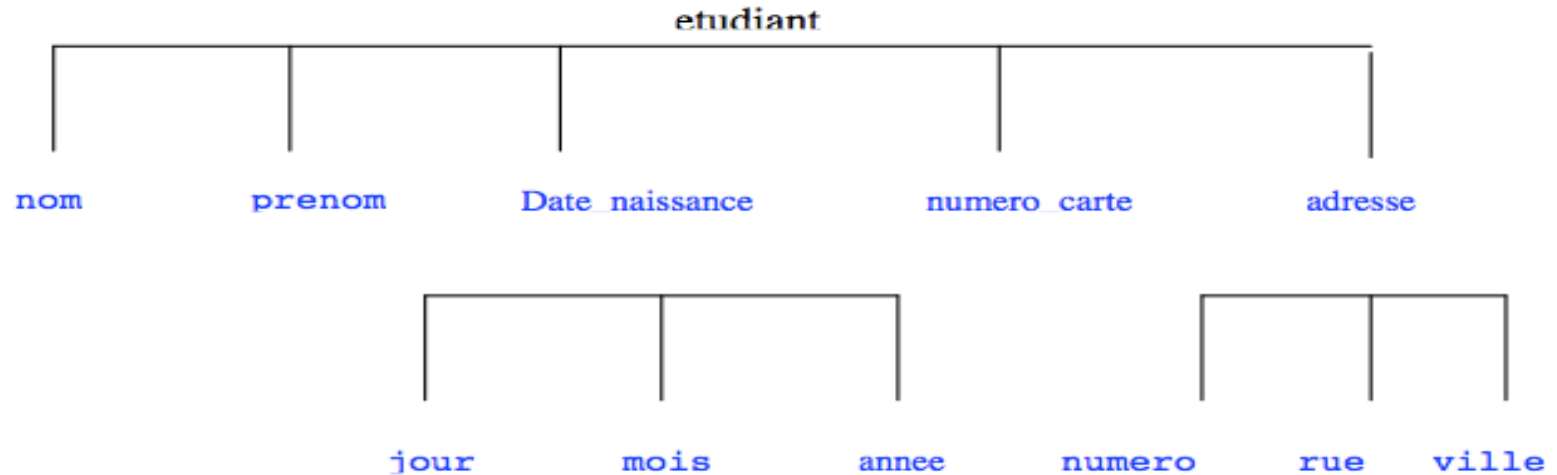
```
    Date_naissance : date ;
```

```
    Adresse       : adr;
```

```
    Numero_carte  : integer;
```

```
END ;
```

Chap8: Les structures d'enregistrement



- L'accès aux champs de l'enregistrement se fait **par niveau** en utilisant toujours l'une des méthodes précédentes (par **nom composé** ou avec l'instruction **with**).
- Ainsi, si on déclare une variable **v_etudiant** de type **etudiant**, on peut accéder au champ jour de la date de naissance comme suit :

```
WRITE(v_etudiant.date_naissance.jour);
```

ou

```
WITH v_etudiant DO
```

```
    WITH date_naissance DO
```

```
        WRITE(jour);
```

Chap8: Les structures d'enregistrement

8.5. Enregistrement avec variantes

- Supposons qu'on veuille ajouter à la description d'un étudiant un champ devant représenter le taux de l'allocation de l'étudiant, et qu'on ait envie d'identifier ce champ par *bourse* s'il a une bourse et par *aide* s'il bénéficie d'une aide.
- Pascal permet de résoudre ce problème en permettant l'inclusion d'une partie variable dans la définition d'un enregistrement. Cette partie doit être placée après la partie fixe, les champs de la partie variable varient selon la valeur d'un champ appelé **sélecteur**. A chaque valeur (ou ensemble de valeurs) possible du sélecteur correspond un ou plusieurs champs dont l'ensemble est appelé **variante**.
- Le sélecteur est considéré comme un champ de l'enregistrement.

Syntaxe :

```
TYPE id_variant = RECORD
    Chp1: type1;
    Chp2 : type2;
    .....
    chpk : typek;
    CASE id_selcteur : typeselecteur OF
        sel1 : (chpk+1: typek+1);
        .....
    END;
END;
```

Chap8: Les structures d'enregistrement

où :

- **id_variant** est l'identificateur du type enregistrement.
- **Chp1,...,chpk** : champ de la partie fixe avec leurs types respectifs **type1,...,typek**
- **Id_sélecteur** et **type sélecteur** : identificateur et type de sélecteur de la partie variable
- **Sel1,...,seln** : les valeurs possibles du sélecteur.
- **Chpk+1,...,chpn** et **typek+1,...,typen**: les identificateurs des champs variables et leurs types respectifs

Exemple:

```
TYPE etudiant_allocation = RECORD
    nom, prenom : string[32];
    date_naissance : string[8];
    adresse      string[60];
    CASE allocation : integer OF
        1 : (bourse: integer);
        2 : (aide : integer);
    END ;
```

Chap8: Les structures d'enregistrement

Remarques:

1. Une variante peut elle-même contenir une partie finale variable. Par exemple, si on veut, dans le cas d'un étudiant boursier, identifier différemment le champ relatif au taux de la bourse selon que c'est une bourse entière ou une demi-bourse.

TYPE etudiant_allocation = **RECORD**

Nom, prenom : string[32];

Date_naiss : string[8];

Adresse: string[60];

Case allocation : integer **of**

1 : (**case** b_e : boolean **of**

true : (bourse : integer);

false : (demi_bourse : integer) ;) ;

2 : (aide : integer);

END;

2. Un seul **END** suffit pour fermer le **RECORD**, et le(s) **CASE**.

Chap8: Les structures d'enregistrement

8.6. Opérations permises sur les variables enregistrement

- **l'affectation:** il est possible d'accéder à tout un enregistrement par cette opération.
 - Par exemple, l'instruction suivante **copie** tout l'enregistrement (tous les champs) **idv2** dans **idv1** où **idv1** et **idv2** sont des variables enregistrement ayant le même type :

idv1 := idv2

- **Constantes d'enregistrement:**

Syntaxe :

CONST<idconst> :<id_type>=(<id_chp>:<valeur> {;<id_chp>:<valeur>}) ;

Lorsqu'il s'agit d'une constante de tableau d'enregistrement, il faudra ajouter des parenthèses (une ouvrante et une fermante).

CONST<idconst> :<id_type>=((<id_chp>:<valeur> {;<id_chp>:<valeur>}),
(<id_chp>:<valeur> {;<id_chp>:<valeur>}),
.....
.....

Chap8: Les structures d'enregistrement

Exemple:

TYPE

Point = Record

X : integer;

Y :integer;

End;

Vecteur = Array [0..1] Of Point;

Mois = (jan, fev, mar, avr, mai, juin, juil, août, sept, oct, nov, dec) ;

Date = record

Jr : 1..31;

Ms : Mois;

An : 1972..1983 ;

End ;

Remarques:

1. Les opérations de lecture, d'écriture et de comparaison avec les opérateurs relationnels $<>$, $>=$, $<=$, $=$ ne sont pas permises. Elles ne peuvent être effectuées que sur les champs.
2. Les seules expressions d'un type enregistrement sont les variables de ce type.
3. Il n'y a pas de fonction (même prédéfini) dont le résultat est de type enregistrement

CONST

Origine : Point=(x :2.0 ; y :2.0);

Ligne : vecteur =((x :-3.1 ; y :1.5), (x :5.8 ; y :3.0)) ;

Jour : date=(jr:2 ;ms :dec ;an :1980) ;

Chap8: Les structures d'enregistrement

8.7. Tableau d'enregistrements

On peut définir des tableaux dont les éléments sont des enregistrements de même type.

Exemple:

Si on veut gérer un parking de voiture, on peut utiliser un tableau composé d'enregistrement de type voiture.

```
CONST nmax = 100 ;  
TYPE voiture = RECORD  
    marque : string[25] ;  
    date_fabrication : string[8] ;  
    immat : string[8] ;  
    nbreplace : integer ;  
END;
```

```
Parking = ARRAY [1..nmax] OF voiture;  
VAR t : Parking;
```

La saisie des éléments de **t** pourra se faire comme suit :

```
FOR I:=1 TO nmax DO  
BEGIN  
    READLN(t[i].marque) ;  
    READLN(t[i].date_fabrication) ;  
    READLN(t[i].immat) ;  
    READLN(t[i].nbplace) ;  
END;
```

Le tableau **t** précédent pouvait être défini comme suit :

```
VAR t : ARRAY [1..nmax] OF RECORD  
    marque : string[25] ;  
    date_fabrication : string[8] ;  
    immat : string[8] ;  
    nbreplace : integer ;  
END;
```

Dr Ousmane Diallo UFR-ST / Dept Info

Chap8: Les structures d'enregistrement

Exercice d'application:

Écrire une **procédure** permettant d'afficher les éléments de **t** et une **fonction** qui renvoie l'indice de l'élément ayant le plus grand nombre de places.

Chap8: Les structures d'enregistrement

Références....



Chap9: Les fichiers

9.1. Généralités

- Les types de données étudiés jusqu'ici servaient à décrire des **informations**. Ces informations étaient **toujours perdues quand nous éteignons l'ordinateur**.
- Par exemple si on utilise un tableau d'enregistrements d'étudiants pour gérer la scolarité, il **faut à chaque exécution saisir les éléments du tableau**.
- Dans ce chapitre, nous allons étudier comment remédier à cet inconvénient en créant et en utilisant les **fichiers**.

9.1.1. Définition

- Un fichier est une **suite éventuellement vide de composants de même type stockés sur une mémoire secondaire** (disquette, disque dur, CD, etc.).
- Il existe en **permanence**, il est **indépendant de tout traitement** et est **accessible par un ou plusieurs programmes**. Il permet :
 - de **sauvegarder l'information** entre plusieurs exécutions,
 - de faire **communiquer** entre eux plusieurs programmes, qu'ils soient sur une machine ou sur des machines différentes,
 - de **transmettre des données** entre plusieurs machines différentes.

Chap9: Les fichiers

9.1.2. Les fichiers séquentiels

Un fichier à **organisation séquentielle** est composé d'articles (éléments) consécutifs sur la mémoire de masse. Ces articles **sont rangés dans l'ordre de leur enregistrement**. Un fichier séquentiel est **facile à implémenter** mais présente quelques inconvénients:

- l'accès est lent pour de grands fichiers,
- l'ajout d'un article n'est possible qu'en fin de fichier.

Les fichiers **en pascal** ont une structure séquentielle.

9.1.2.1. Organisation et Accès

L'organisation définit la manière dont les articles sont disposés sur le support. On distingue **trois types d'organisations**:

- l'**organisation séquentielle** qui ne permet que l'**accès séquentiel**. On accède au fichier dans l'ordre d'enregistrement des articles,
- l'**organisation directe** qui permet l'**accès direct**. On accède directement à un élément sans passer par les articles précédents,
- l'**organisation séquentielle indexée** qui permet l'**accès séquentiel** et l'**accès direct**.

Chap9: Les fichiers

9.1.2.2. Caractérisations générales d'un fichier

Un fichier est caractérisé par:

- le nom,
- l'unité d'information,
- l'organisation,
- le support.

Le nom du fichier permet de l'**identifier**, on l'appelle **nom externe**. Il est formé selon les règles du système d'exploitation. Parfois il faut faire précéder ce nom du chemin d'accès.



9.1.2.3. Traitement des fichiers

A. Association entre une variable fichier et un fichier externe

- Pour pouvoir être utilisée, une **variable fichier** ou **fichier interne** (située en mémoire vive) doit impérativement être initialisée en l'associant à un **fichier externe**.
- Toutes les opérations sur le **fichier interne** affecteront le **fichier externe**. On parle de **fichier disque** (le **fichier externe**) et de **fichier logique** (le **fichier interne**).

Chap9: Les fichiers

B. Ouverture d'un fichier

- Avant toute utilisation, il faut avoir le fichier ouvert. L'ouverture du fichier permet au système d'exploitation la création d'une **zone de transfert en mémoire**, le **tampon d'échange**.
- En général, on distingue trois modes d'ouverture d'un fichier :
 - ouverture en **lecture seule**,

 - ouverture en **écriture seule**,

 - ouverture en **lecture / écriture**.
- Selon le mode d'ouverture l'exploitation du tampon sera différente. L'information n'est pas immédiatement déposée sur le disque après l'opération d'écriture ni qu'elle vient du disque après opération de lecture.

C. Fermeture d'un fichier

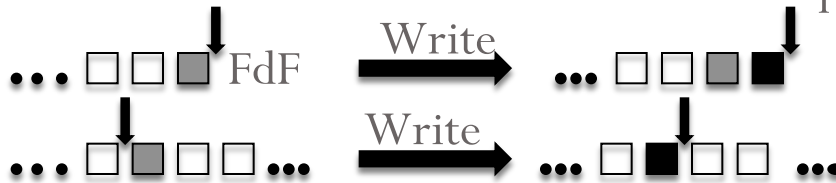
- En fin d'utilisation, un fichier doit être fermé pour une **purge éventuelle du tampon interne**.
- Ce n'est qu'après cette opération que l'on peut être assuré que toutes les opérations d'écriture sont **physiquement réalisées** sur le fichier disque ou fichier externe.

Chap9: Les fichiers

D. Ecriture et Lecture dans un fichier

Lorsqu'un fichier est ouvert en écriture, on peut l'agrandir en effectuant une opération d'écriture. Eventuellement, on peut modifier ou supprimer des données du fichier.

- Ecriture: écrire l'unité au niveau du repère et déplacer le repère



Lorsqu'un fichier est ouvert en lecture ou lecture / écriture, on peut lire, extraire des informations.

- Lecture: lire l'unité au niveau du repère et déplacer le repère



9.2. Les fichiers en Pascal

En pascal, la structure d'un fichier est séquentielle. Les fichiers sont au nombre de trois:

- Les fichiers typés,
- Les fichiers non typés,
- Les fichiers texte

Chap9: Les fichiers

- Les fichiers typés et non typés sont des fichiers binaires.
- Le nombre de composants d'un fichier c'est à dire la **taille du fichier n'est pas prédéterminée**. Il est limité par la taille disponible sur le support.
- En pascal, chaque fois qu'un composant est écrit ou lu dans un fichier, le pointeur de fichier est automatiquement avancé sur le composant suivant.

9.2.1. Les fichiers typés

9.2.1.1 Déclaration

A. En pseudo-code

TYPE identificateur_du_type_du_fichier = **FICHIER** identificateur_du_type_des_elements;

VARIABLE identificateur_de_la_variable : identificateur_du_type_du_fichier;
 identificateur_de_la_variable : **FICHIER** identificateur_du_type_des_elements ;

B. En pascal

TYPE identificateur_du_type_du_fichier = **FILE OF** identificateur_du_type_des_elements ;

VAR identificateur_de_la_variable : identificateur_du_type_du_fichier ;
 identificateur_de_la_variable : **FILE OF** identificateur_du_type_des_elements ;

- Un fichier typé en pascal est défini par le mot réservé **FILE** suivi par le type des composants de ce fichier. **Le type des composants d'un fichier peut être quelconque, sauf un type fichier.**

Chap9: Les fichiers

Exemples

TYPE

```
fic = file of integer ;  
fic1 = file of real;  
enr = record  
    nom, prenom:string[20];  
    numero_carte:integer;  
end;  
fichier_etudiant = file of enr;
```

VAR

```
f1: fic;  
f2: fic1;  
etudiant : fichier_etudiant;  
etud : file of enr;
```

9.2.1.2 Association entre le fichier interne et le fichier externe

L'association permet de faire le lien logique entre le fichier interne et le fichier externe. Elle est réalisée avec la primitive suivante:

Chap9: Les fichiers

A. en pseudo-code

ASSOCIER(f, nom_fic)

B. en pascal

ASSIGN(f, nom_fic)

La variable **f** est une variable de type fichier (**file of**) et **nom_fic** est une **constante chaîne de caractères** ou une **variable de type chaîne de caractères**. **f** représente le **fichier logique** et **nom_fic** est le **fichier disque**.

Exemple 1

- Le fichier **essai.dat** se trouve dans le **lecteur C** dans le sous répertoire **travail** du répertoire **TP**.
- Le fichier **essai** se trouve dans la **racine du lecteur C**.

en pseudo-code:

VARIABLES

f: FICHER DE réels

g: FICHER D'entiers

Debut

ASSOCIER (f, 'c:\tp\travail\essai.dat')

ASSOCIER(g, 'essai') .

... {traitement sur les fichiers}

Fin.

Chap9: Les fichiers

en Pascal:

```
VAR
    f: FILE OF real ;
    g: FILE OF integer;
Begin
    ASSIGN(f, 'c:\tp\travail\essai.dat') ;
    ASSIGN(g, 'essai') ;.
    ... {traitement sur le fichier}
End.
```

Exemple 2

On définit d'abord la constante **nom_fic**, où on met le chemin d'accès du fichier. On appelle la procédure **ASSIGN**.

```
CONST  nom_fic = 'c:\tp\travail\donnees.dat' ;
VAR     F : FILE OF ... ;
Begin
    ASSIGN(F, nom_fic) ;
    ... {traitement sur le fichier}
End.
```

Chap9: Les fichiers

Exemple 3

On saisit le nom externe du fichier avant de faire l'assignation.

```
VAR nom_fic : string;
```

```
    F: FILE of ...;
```

```
Begin
```

```
WRITE('Sur quel fichier va-t-on travailler ?') ;
```

```
    READLN(nom_fic) ;
```

```
    ASSIGN(F, nom_fic) ;
```

```
    ... {traitement sur le fichier}
```

```
End.
```

9.2.1.3. Ouverture d'un fichier

Avant de pouvoir être exploitée, une variable fichier doit être associée à un fichier externe, puis le fichier doit être ouvert par l'une des méthodes suivantes :

A.ouverture en écriture seule

en pseudo-code

```
    ouvrir(f :fichier)
```

en pascal

```
    REWRITE (f) ;
```

où **f** est une **variable fichier**.

Chap9: Les fichiers

- Un nouveau **fichier disque** (fichier externe) dont le nom a été préalablement assigné à la variable **F** est **créé et préparé pour le traitement en écriture seule**.
- Le contenu de tout fichier préexistant avec le même nom est **détruit**. Un fichier disque créé par **REWRITE** ne **contient aucune information** ;

B. ouverture en lecture seule ou en lecture écriture
en pseudo-code

ouvrir(f: fichier)

en pascal

RESET(f) ;

- Le fichier disque dont le nom est préalablement assigné à la variable **F** est préparé pour le traitement en **lecture seule ou écriture**. Le **pointeur fichier est positionné au début de ce fichier**.
- Le **nom du fichier externe correspondant doit exister sur le disque**, sinon il se produira une **erreur d'entrée/sortie**.

Chap9: Les fichiers

9.2.1.4. Fermeture d'un fichier

L'unique procédure disponible pour fermer un fichier est la suivante :

en pseudo-code

fermer(f : fichier)

en pascal

CLOSE(f) ;

La variable interne redevient **disponible** pour une autre association ou ouverture du même fichier externe dans un autre mode.

Exemple

Les instructions ci-dessous créent un fichier de **taille zéro (vide)** nommé **essai** et situé sur le lecteur a :

...

ASSIGN(f, 'a:\essai') ;

REWRITE(f); {creation}

{traitement en écriture seule}

CLOSE(f);

...

Chap9: Les fichiers

Les instructions suivantes **ouvrent en lecture/écriture** le fichier **essai** :

...

ASSIGN(f, 'a:\essai') ;

RESET(f);

{traitement en lecture/écriture seule}

CLOSE(f);

...

9.2.1.5. Ecriture dans fichier

- Pour écrire dans un **fichier typé**, il faut **l'ouvrir en écriture seule ou en lecture/écriture**.
- L'écriture dans le fichier permet d'**agrandir la taille du fichier**. Cette opération s'effectue avec la primitive suivante:

en pseudo-code

ecrire(f, v_article)

en pascal

write(f,v_article) ; {permet d'écrire v_article à la position courante dans le fichier. }

où **f** est le fichier logique et **v_article** la **variable contenant l'article de même type que les éléments du fichier**.

Chap9: Les fichiers

Exemple 1

On veut créer un répertoire téléphonique, un fichier contenant l'ensemble des personnes ainsi que leurs numéros de téléphone.

en pseudo-code

TYPE

personne = **ENREGISTREMENT**
 nom : chaîne de 32 caractères
 prenom: chaîne de 32 caractères
 numtel : chaîne de 10 caractères

FIN DE L'ENREGISTREMENT

Repertoire_telephonique = **FICHER DE** personne

VARIABLE

fpers : Repertoire_telephonique
 vpers :personne

DEBUT

{on veut écrire l'article Badou SAMB 77 961 19 06 dans le fichier.}

ASSOCIER(fpers, 'repertoire')

OUVRIER(fpers) *{création, ouverture en écriture seule}*

vpers.nom ← 'DIOP'

vpers.prenom ← 'Bouba'

vpers.numtel ← '77 961 19 06'

ECRIRE(fpers, vpers)

Fermer(fpers)

FIN

Chap9: Les fichiers

en pascal

TYPE

personne = **RECORD**

nom : string[32] ;

prenom: string[32] ;

numtel : string[10] ;

END;

Repertoire_telephonique = **FILE OF** personne;

VAR

fpers : Repertoire_telephonique;

vpers : personne;

BEGIN

{on veut écrire l'article Badou SAMB 77 961 19 06 dans le fichier.}

ASSIGN(fpers, 'repertoire');

REWRITE(fpers); *{création, ouverture en écriture seule}*

vpers.nom := 'DIOP';

vpers.prenom := 'Bouba';

vpers.numtel := '77 961 19 06';

WRITE(fpers, vpers);

CLOSE(fpers);

END.

Chap9: Les fichiers

Exemple 2

On veut créer un fichier d'étudiants.

```
TYPE Etud = RECORD
    nom, prenom :string[32] ;
    date_naiss : string[10];
    num_ins : string[12];
END ;

VAR
    Etudiant : etud;
    Fetud : FILE OF etud;
    i, n :integer ;

BEGIN
    ASSIGN(Fetud,'scolaire');
    REWRITE(Fetud);
    WRITE('Combien d"étudiants voulez vous saisir ?');
    READLN(n);
```

```
FOR i =1 TO n DO
    BEGIN
        WITH Etudiant DO
            BEGIN
                WRITE('Numéro d'inscription :');
                READLN (num_ins);
                WRITE ('Nom:');
                READLN(nom);
                WRITE ('Prénom :');
                READLN (prenom);
                WRITE ('Date de naissance :');
                READLN (date_naiss);
            END;
            WRITE(Fetud, Etudiant);
        END;
    CLOSE(Fetud);
END.
```

Chap9: Les fichiers

9.1.6. Lecture dans un fichier

Pour lire dans un **fichier typé**, il faut l'ouvrir en **lecture seule** ou en **lecture écriture**. Cette opération s'effectue avec la primitive suivante:

en pseudo-code

lire(f, v_article)

en pascal

read(f, v_article) ; *{permet de lire dans v_article la donnée située à la position courante de lecture.}*

où **f** est le **fichier logique** et **v_article** la variable contenant l'article de même type que les éléments du fichier.

Exemple:

On veut afficher les personnes du répertoire téléphonique, i.e. lire les enregistrements du fichier.

en pseudo-code

DEBUT

ASSOCIER(fpers, 'repertoire')

OUVRIIR(fpers) *{lecture, ouverture en écriture seule}*

LIRE(fpers, vpers)

ECRIRE(vpers.nom)

ECRIRE (vpers.prenom)

ECRIRE (vpers.numtel)

Fermer(fpers)

FIN

en Pascal

BEGIN

ASSIGN(fpers, 'repertoire');

RESET(fpers) ;

READ(fpers, vpers);

WRITE('nom :', Verps.nom);

WRITE('prenom :', Verps.prenom);

WRITE('numero de telephone :', Verps.numtel);

CLOSE(fpers);

END.

Chap9: Les fichiers

Exercice d'application:

Afficher le contenu du fichier étudiant.

Remarque

- L'opération de lecture ne peut s'exécuter que si le fichier **n'est pas vide** et, s'il n'est pas vide, il faut que le dispositif de lecture / écriture (le **pointeur**) **ne soit pas à la fin de fichier**.

9.2. Fonctions de manipulation des fichiers typés

9.2.1. La fonction fin de fichier

- Elle permet de tester si le dispositif de lecture / écriture est à la fin du fichier. Elle retourne vrai si le pointeur de fichier est positionné après la dernière composante d'un fichier.
- L'opération s'effectue avec la primitive suivante:

EOF(f) ;

- Supposons que le fichier disque **scolaire** contient plusieurs articles et qu'on ignore le nombre d'articles, si on doit lire et afficher tous les articles du fichier, il faut **avant chaque lecture tester si la fin de fichier est atteinte**.

Chap9: Les fichiers

```
BEGIN
  ASSIGN(fetud,'scolaire');
  RESET(fetud);
  WHILE NOT EOF(fetud) DO
    WITH etudiant DO
      BEGIN
        READ(fetud, etudiant);
        WRITE('Numéro d'inscription :',num_ins);
        WRITE('Nom:',nom) ;
        WRITE('Prénom :',prenom) ;
        WRITE('Date de naissance :',date_naiss);
      END;
    CLOSE(fetud);
  END.
```

9.2.2 La fonction IORESULT

- Elle permet au programmeur de contrôler les éventuelles **erreurs d'entrée / sortie (E/S)** qui peuvent subvenir lors de l'exécution d'un programme.
- Par exemple, la **tentative d'ouverture en lecture ou en lecture / écriture d'un fichier inexistant** ou la **tentative d'écriture sur un fichier disque plein** provoquent une erreur E/S et une interruption brutale de l'exécution du programme en cours.

Chap9: Les fichiers

- Pour éviter ce désagrément, nous devons utiliser les directives de compilation `{ $I- }` et `{ $I+ }`.
- **Lorsque le contrôle d'entrées-sorties est désactivé** (lorsqu'une série d'instructions est précédée de `{ $I- }`), **une erreur E/S ne cause pas l'arrêt brutal du programme, mais suspend toute autre opération E/S jusqu'à ce que la fonction IORESULT soit appelée.**
- Après chaque opération E/S, nous testons la fonction IORESULT: **elle renvoie 0 en cas de succès et un entier différent de 0 sinon.**

Exemple:

Ecrire un programme qui ouvre en lecture / écriture un fichier de réels dont le nom est saisi au clavier en vérifiant d'abord son existence.

```
VAR
  f : FILE OF REAL;
  nomfic : STRING[11];
BEGIN
  WRITELN('Donner le nom du fichier');
  READLN(nomfic);
  ASSIGN(f, nomfic);
  { $I- } { désactive l'option de détection des erreurs d'E/S }
  RESET(f); { l'exécution est arrêtée si f n'existe pas et l'option activée }
  { $I+ } { réactive l'option de détection des erreurs d'E/S }
```

```
IF IORESULT <> 0 THEN
  Write('Fichier inexistant !')
Else
  { traitement du fichier }
  ...
CLOSE(f);
END.
```


Chap9: Les fichiers

9.2.3 La fonction FILEPOS

- C'est une fonction entière qui retourne la valeur actuelle du pointeur de fichier.
- La position est donnée sous forme d'un entier compris entre 0 et le nombre d'enregistrements – 1 (filesize(f)-1).
- Avec un fichier qui vient d'être ouvert la valeur est zéro.

FILEPOS(f)

9.2.4 La fonction FILESIZE

- C'est une fonction qui retourne la taille d'un fichier disque exprimée en nombre d'articles.
- Elle correspond donc à la valeur maximale que l'on peut affecter au pointeur de fichier.
- Si FILESIZE est égal à 0 le fichier est vide. Si f est un fichier vide alors FILEPOS(f) est égale à FILESIZE(f).

9.2.5 La procédure SEEK

- Elle permet de déplacer le pointeur de fichier sur le **n^{ième}** article du fichier.
- n est une expression entière de type long (**LongInt**).
- Avec cette procédure on peut aller directement à n'importe quel enregistrement du fichier et il n'a pas besoin de parcourir les enregistrements précédents.
- La syntaxe est la suivante :

Dr Ousmane Diallo UFR-ST / Dept Info
SEEK(f , numero_enregistrement) ;

Chap9: Les fichiers

- La position de la première fiche est **0**.
- L'instruction **SEEK(f,0)** permet de se déplacer au **début du fichier**.
- Pour agrandir un fichier on doit déplacer le pointeur juste après le dernier enregistrement et écrire à cet endroit. On utilise l'instruction suivante:

SEEK(f, FILESIZE(f)) ;

9.2.6 La procédure TRUNCATE

Elle coupe le fichier à la position courante ; il s'agit de l'unique moyen de réduire la taille d'un fichier.

9.3. Mise à jour d'un fichier

La mise à jour consiste à:

- Créer de nouveaux articles,
- Modifier des articles,
- Supprimer des articles.
- Toute opération **de consultation ou de mise à jour doit commencer par un travail de recherche**, qui consiste à trouver l'élément que l'on veut consulter, modifier, ou supprimer.
- La recherche est effectuée sur un **champ de l'enregistrement**.
- Dans l'exemple qui suit, on effectue une recherche selon **le numéro d'inscription** qui est unique. Le programme traite un fichier qui permet de gérer une amicale d'étudiants.

Dr Ousmane Diallo UFR-ST / Dept Info

Chap9: Les fichiers

```

program gestion_amical ;
TYPE
    adherent = record
        num_ins : string[12];
        nom, prenom : string[25] ;
        cod_postal : integer;
        num_tel : string[10];
    end ;
VAR
    membre : adherent ;
    amical : File of adherent ;
    numero :String[12];
    trouve :boolean;
BEGIN
    ASSIGN(amical, 'c : \amical\club').
    {$I-} RESET (amical); {$I+}
    IF IORESULT <> 0 THEN
        WRITELN(' le fichier n'existe pas')
    ELSE
        IF FILESIZE(amical) = 0 THEN
            WITELN('le fichier est vide')
        ELSE

```

```

BEGIN
    trouve := FALSE ;
    WRITELN('taper le numéro à saisir');
    READLN(numero) ;
    WHILE NOT EOF(amical) DO
        BEGIN
            READ(amical, membre);
            IF member.num_ins = numero THEN
                trouve := TRUE ;
            END ;
            IF trouve THEN
                WRITELN(nom,' ', prenom, ' ', code_postal,' ', num_tel )
            ELSE
                WRITELN('le numéro n'exsite pas dans le fichier') ;
            CLOSE(amical);
        End;
    END.

```

Chap9: Les fichiers

9.4. Les fichiers texte

9.4.1. Définition

- Un fichier texte est un fichier de caractères disposés en ligne de texte.
- En turbo pascal, on dispose du type prédéfini **TEXT** qui permet de déclarer des fichiers texte.

VAR

identificateur_de_la_variable : **TEXT**;

Exemple:

VAR f1, fic, f2: **TEXT**;

- L'association entre la variable fichier et le fichier disque se fait de la même manière que pour les fichiers typés.

9.4.2 Ouverture et fermeture d'un fichier texte

- Avec les fichiers texte, on dispose de trois modes d'ouverture :
 - en écriture seule (la création),
 - en écriture seule (ajout d'éléments)
 - et en lecture seule.
- L'ouverture en **lecture seule** et l'ouverture en **écriture seule (nouveau fichier)** s'effectuent de la même manière en appelant respectivement la primitive **RESET** et la primitive **REWRITE**.

Chap9: Les fichiers

- Pour ouvrir un fichier texte existant en **mode ajout**, la procédure **APPEND** doit être utilisée à la place de **REWRITE**.

```
{ $I- }
APPEND(T) ;
IF IORESULT <> 0 THEN
REWRITE(T);
{ $I+ }
```

- La fermeture se fait avec la primitive **CLOSE**.

9.4.3. Fonctions de manipulation des fichiers texte

- Fonction **EOF**(var T :TEXT)
- Cette fonction permet de tester si le pointeur est positionné sur la marque de fin de fichier.
- **EOF** est **toujours vrai** pour un fichier ouvert avec **REWRITE** et **APPEND**.
 - Fonction **SEEKEOF**(var T :TEXT) : boolean
- Similaire à EOF, **mais élimine les espaces**, les **tabulations**, et les **lignes vides** avant de tester la marque de fin de fichier.
 - Fonction **EOLN**(var T :TEXT) : boolean
- C'est une fonction booléenne retournant **VRAI** si la fin de ligne a été atteinte.
- Si EOF(T) est vrai alors EOLN(T) est également vrai.

Chap9: Les fichiers

- Fonction **SEEKLN**(var T :TEXT) :boolean
- Similaire à **EOLN**, mais **écarte les espaces, les tabulations**, et les **lignes vides** avant de tester la marque de fin de ligne.
 - Les fichiers texte utilisent les primitives **READ**, **READLN**, **WRITE**, **WRITELN** exactement comme pour les entrées sorties au clavier et à l'écran.
- La procédure **READ**(var T :TEXT, V1, V2, ..., Vn);
 - ✓ Elle permet l'entrée de caractères, chaînes de caractères, et de données numériques.
- La procédure **READLN**
 - ✓ Elle identique à **READ**, excepté que lorsque la dernière variable à été lue, le reste de la ligne est ignoré.