Les Fichiers en Pascal

Université de Toulouse II

DEUG MASS

Année 2004-2005

Patricia PASCAL

< ppascal@laas.fr >

Plan du cours sur les fichiers

Documents additionnels :

Langage Pascal *par Pierre Breguet*

`http://ina.eivd.ch/ina/publications/coursPascal/welcome.htm`

Introduction

- programme Pascal = données en mémoire physique

  ces données sont volatiles ⇒ risques ?

  ces données ne sont pas persistantes ⇒ problèmes ?

- nécessité d'un moyen de stockage permanent :

  Disque dur, disquettes, Cdroms …

- notion de fichier :

  enregistrement de données sur support physique permanent

  ⇒ un fichier :

  - un identifiant (nom)
  - un emplacement (lié au support)
  - un contenu (données)

Précisions sur la notion de fichier

- la notion d'identifiant de fichier

  nom externe : c'est le nom du fichier pour le système de fichier du système d'exploitation considéré, par exemple : mon_fichier.dat

  nom interne : c'est le nom du fichier connu par le programme, il est déclaré par le programme, par exemple : Fichier_de_données

  $\Rightarrow$ nécessité d'associer le fichier physique à sa représentation interne : commande ***Assign***

- le « repère » de position

  Il permet de savoir où l'on se trouve dans un fichier lorsque l'on accède à celui-ci dans l'ordre de ses enregistrements (acœs séquentiel).

  C'est aussi lui qui détermine le résultat des opérations telles que : Eof ou Eoln.

Différents types de fichiers

- Un fichier informatique est organisé en ***enregistrements***, chaque enregistrement contenant une collection d'unités logiques d'informations encore appelées rubriques. Souvent les enregistrements ont la même structure.

- Une ***rubrique*** est la plus petite unité logique d'information ayant un sens en tant que tel. La taille d'une rubrique s'exprime en nombre de caractères et peut être fixe ou variable.

- La façon selon laquelle ces rubriques et enregistrements sont organisés dans le fichier détermine le type de fichier :
  - ordre physique = ordre logique : *organisation* ***séquentielle*** (naturelle à Pascal)
  - enregistrements de taille fixe, portant un numéro qui est relatif au début du fichier : *organisation* ***relative***
  - chaque enregistrement est associé à une clé : *organisation* ***indexée***

Méthodes d'accès aux données

- Les méthodes par lesquelles on lit ou on écrit un enregistrement d'un fichier sont appelées les ***méthodes d'accès***.

- La ***méthode d'accès*** que l'on veut utiliser doit être spécifiée ***au moment de l'ouverture du fichier***. Un même fichier peut être accédé par des méthodes différentes selon son ***organisation*** qui elle a été définie ***au moment de sa création***.

- Les différentes méthodes d'accès :
  - ***accès séquentiel*** : enregistrements traités en séquence
  - ***accès direct*** : accès direct par le numéro d'enregistrement
  - ***accès indexé*** : accès par l'ordre des clés d'accès

Rq : on ne peut utiliser que la méthode d'accès séquentielle avec une organisation de fichier séquentielle (Turbo Pascal permet également la méthode d'accès direct)

Rappels de typage en Pascal : enregistrements

- **Rappel :**

  Un enregistrement est un type complexe, composé (généralement) de plusieurs champs ou rubriques. Cela permet de définir un élément contenant plusieurs informations de types éventuellement différents.

- **Déclaration d'un enregistrement :**

```
type t_nombre_complexe = record (* nombre complexe *)
                                  partie_reelle : real;
                                  partie_imaginaire : real;
                                 end;
var nombre_1,
    nombre_2 : t_nombre_complexe;
```

- **accès aux champs d'un enregistrement :**

```
nombre_1.partie_reelle         (* accès au premier champ *)
nombre_2.partie_imaginaire     (* accès au deuxième champ *)
```

Rappels sur la gestion du type string

- **Déclaration d'une chaîne de caractères :**

```
const   max = 10;                          (* taille du type chaîne *)
type    t_chaine = string [  max ];        (* déclaration du type *)
var     chaine_1, chaine_2 :t_chaine;      (* déclaration des variables *)
```

- **Opérations principales sur une chaîne :**

```
chaine_1 := 'test';    (* affectation d'un chaîne constante à une
    variable de type chaîne, attention il faut length_1 >= length_2 *)
length ( < chaîne > ), (* donne la longueur réelle de chaîne *)
read ( < chaîne > );    (* lit une chaîne au clavier *)
readln ( < chaîne > ); (* même opération et lit le caractère /eoln *)
write ( < chaîne > );   (* écrit une chaîne à l'écran *)
writeln ( < chaîne > );(* même opération et écrit le caractère /eoln *)
```

- **Remarques additionnelles sur les chaînes :**
  - ✓ une chaîne ne peut pas dépasser 255 caractères
  - ✓ un caractère est une chaîne de taille 1
  - ✓ la chaîne vide existe : ''

Fichiers binaires et fichiers de texte

❑ Un fichier séquentiel sera donc structuré selon le type de données qu'il contient.

⇒ données de type enregistrement :

fichier binaire (contient la structure de l'enregistrement)

⇒ données de type caractère :

fichier texte

Utilisation des fichiers d'enregistrements (1)

Déclaration et lien physique d'un fichier

• **Déclaration :**

```
type     nombre_complexe = record (* nombre complexe *)
                             partie_reelle : real;
                             partie_imaginaire : real;
                           end;
         fichier_entiers = file of integer;
         fichier_nombres_complexes = file of nombre_complexe;

var      int_1, int_2 : integer;
         nombre_1,nombre_2 : nombre_complexe;
         fichier_int : fichier_entiers;
         fichier_complexe : fichier_nombres_complexes;
```

Rq : il existe deux variables fichiers prédéfinies, input et output de type text qui n'ont pas besoin d'être déclarées

• **Assignation physique :**

```
assign ( < File >, < nom du fichier réel >);
```

Utilisation des fichiers d'enregistrements (2)

Principales commandes sur les fichiers

- **Indicateur de fin de fichier :**

```
Eof ( < File > ) ;
```

(Valeur booléenne, `True / False`)

- **Exemples d'associations par `assign` :**

```
var Destination: String ;
FOrigine, FDestination: FILE OF Integer ;
assign ( FOrigine, 'C:\DATA\Infile.Dat' );
readln ( Destination );
assign ( FDestination, Destination );
```

- **Opérations d'ouverture / fermeture :**

```
reset ( < File > );      (* ouverture du fichier en lecture  *)
rewrite ( < File > );    (* ouverture en écriture / création  *)
close ( < File > );      (* fermeture du fichier, important !! *)
```

Utilisation des fichiers d'enregistrements (3)

Principales commandes sur les fichiers (suite)

- **Opérations de lecture / écriture (Pascal):**

```
livre := bibliotheque^;          (* copie la valeur de l'élément
                                    courant dans livre *)
get ( bibliotheque );            (* positionne la fenêtre sur
                                    l'élément suivant *)

bibliotheque^ := livre;          (* copie la valeur de livre dans la
                                    fenêtre *)
put ( bibliotheque );            (* écrit la valeur contenue dans la
                                    fenêtre, avance celle-ci *)
```

- **Opérations de lecture / écriture (TurboPascal):**

```
read ( < File > , < liste de variables > );
write ( < File > , < liste de variables > );
write (output , < variable > { , < variable > } );
read (input , < variable > { , < variable > } );
```

Exemple d'utilisation d'un fichier typé

Lecture d'un fichier de nombres réels et calcul de leur somme

```
. . .
TYPE FichierSequentiel = FILE OF Real ;
. . .
VAR F: FichierSequentiel;
. . .
FUNCTION SommeFichierReel ( VAR Fichier: FichierSequentiel ): Real
;
  VAR X, S: Real ;
BEGIN
  S:=0;
  WHILE NOT Eof ( Fichier ) DO BEGIN
    Read ( Fichier, X );
    S := S + X
  END ; (* -- WHILE, Eof( Fichier ) *)
  SommeFichierReel := S
END ; (* -- SommeFichierReel  *)
. . .
BEGIN {main}
  Assign ( F, 'C:\REELS.DAT' );
  Reset ( F ); {* ouverture du fichier en mode lecture  *)
  Writeln ( SommeFichierReel( F ) );
  Close ( F )
END . (* -- main  *)
```

Exemple d'utilisation d'un fichier typé (2)

Copie d'un fichier binaire dans un autre

Partie déclarative

```
(* Programme : ...
   Fonction : ...
   Auteur : ... *)
program copie_de_fichiers (input, output, original, copie);

const     long_max_nom = 30; (* nombre maximal de caractères d'un nom *)
          long_max_titre = 80; (* nombre maximal de caractères d'un titre *)

type      t_auteur_livre = string [ long_max_nom ];
          t_titre_livre = string [ long_max_titre ];
          t_livre = record           (* représente une fiche bibliographique *)
            titre : t_titre_libre;       (* titre du livre *)
            auteur : t_auteur_livre; (* nom de l'auteur *)
            ISBN : integer;              (* cote en bibliothèque *)
            annee : integer;             (* année de parution *)
          end;
          t_fichier_biblio = file of t_livre;

var       original : t_fichier_biblio;   (* fichier à copier *)
          copie : t_fichier_biblio;      (* copie à créer *)
```

Exemple d'utilisation d'un fichier typé (2bis)

Programme principal

```
.. .. ..

begin (* copie_de_fichiers *)

  assign (original, 'Original.dat');  (* association au fichier d'origine *)
  assign (copie, 'Copie.dat');        (* association au fichier duplicata *)

  reset (original);                   (* ouvrir le fichier à copier *)
  rewrite (copie);                    (* initialiser la copie *)

  while not eof (original) do         (* parcourir le fichier à copier *)
  begin
    copie^ := original^;              (* copier l'élément courant d'une fenêtre *)
                                      (* dans l'autre *)
    put (copie);                      (* écrire l'élément copié *)
    get (original);                   (* passer à l'élément à copier suivant *)
  end; (* while not eof (original) *)

  close (original); close (copie); (* fermeture en fin de programme *)

end. (* copie_de_fichiers *)
```

Cas particulier : les fichiers de texte

• Les fichiers de texte sont des ***cas particuliers de fichiers***. Un fichier de texte est formé d'éléments bien connus : les caractères. Chacun a déjà manipulé de tels fichiers : un programme (Pascal ou autre) est en fait un fichier de texte!

• Les caractères contenus dans un fichier de texte sont organisés en lignes, chacune terminée par une ***marque de fin de ligne***. Après la dernière ligne, le fichier se termine par une ***marque de fin de fichier***.

• Tout ce qui a été dit est valable pour les fichiers de texte. Précisons simplement qu'un fichier est un fichier de texte s'il est déclaré au moyen du type prédéfini **`text`** .

Rq : Un fichier de type text n'est généralement pas équivalent à un fichier de "type" : `file of char` *qui, lui, ne possède pas une structure de lignes !*

• **Déclaration d'un fichier de texte :**

```
f_text : text;    (* text est un fichier de type texte,
                     mot réservé en Pascal, file of n'est pas
                     nécessaire avec les fichiers textes *)
```

Commandes particulières aux fichiers de texte

- **Commandes de base sur les fichiers :**

  On retrouve toutes les commandes déjà vue pour les fichiers binaires (fichiers d'enregistrements), la spécificité des fichiers de texte permet de rajouter les commandes liées aux *strings*.

- **Indicateur de fin de ligne dans le fichier :**

```
Eoln ( < File > );    (* Valeur booléenne, True / False *)
```

(au clavier, `Eoln` *se fait avec <ret>, et* `Eof` *avec <enter>)*

- **Opérations d'ouverture / fermeture :**

```
read ( < File >, < variable > { , < variable > } );
                        (* lit une chaîne dans le fichier < File > *)
readln ( < File >, < variable > { , < variable > } );
                        (* même opération et lit le caractère /eoln *)
write ( < File >, < variable > { , < variable > } );
                        (* écrit une chaîne dans le fichier < File > *)
writeln ( < File >, < variable > { , < variable > } );
                        (* même opération et écrit le caractère /eoln *)
page ( < File> );       (* écrit une marque de saut de page dans le
                           fichier à ne pas utiliser avec output *)
```

Lecture dans un fichier texte

- **Lecture d'un fichier texte, 2 options**

  Un fichier texte (étant une suite séquentielle de caractères) peut se lire caractère par caractère, ou ligne par ligne.

  Le choix dépendra du programme à réaliser, généralement on lit ligne par ligne.

- **Partie déclarative des deux programmes**

```
exemple : text;                (* fichier texte *)
caractere : char;              (* caractère simple *)
chaine_caracteres : string;    (* chaîne de caractères *)
```

Version lecture caractère par caractère

```
while not eof ( exemple ) do
begin
    while not eoln ( exemple ) do
    begin
        read ( exemple,caractere );
           ... (* traiter le
                  caractère lu *)
      end;
      readln ( exemple );
End;
```

Version lecture ligne par ligne

```
while not eof ( exemple ) do
begin
   readln ( exemple,
      chaine_caracteres );
   ... (* traiter la
          chaîne lue *)
End;
```

Exemple sur les fichiers de texte : copie de fichier

Copie de fichier texte avec lecture par caractère

```
var       f_entree,f_sortie : text;
          ch : char;
begin
  (* assignations *)
  assign (f_entree,'In.dat');
  assign (f_sortie,'Out.dat');
  (* ouvertures *)
  reset (f_entree);                    (* fichier ouvert en lecture *)
  rewrite (f_sortie);                  (* fichier ouvert en écriture *)
  while not eof (f_entree) do          (* boucle sur le fichier *)
  begin
      while not eoln (f_entree) do     (* boucle sur la ligne *)
      begin
          read (f_entree,ch);          (* lecture d'un caractère *)
          write (f_sortie,ch);         (* écriture d'un caractère *)
      end;
  readln (f_entree);                   (* passage à la ligne, entrée *)
  writeln (f_sortie);                  (* passage à la ligne, sortie *)
  end;
  (* fermeture *)
  close (f_entree);
close (f_sortie); (* fermeture des 2 fichiers *)
end.
```

Compléments : fichiers externes / internes

• Les ***fichiers externes*** (ou permanents) subsistent après la fin de tout programme les utilisant. Un fichier est rendu externe s'il **est** mentionné comme paramètre du programme au même titre que *input* et *output*.

Exemple:

```
program copie_de_fichiers (input, output, original, copie);
...
   type     t_fichier = file of ...;
   var      original : t_fichier;      (* deux fichiers externes *)
            copie : t_fichier;
...
```

• Les ***fichiers internes*** (ou temporaires) sont détruits automatiquement à la fin du programme. Un fichier est rendu interne s'il **n'est pas** mentionné comme paramètre du programme.

Exemple:

```
program fichiers_internes (input, output);
...
   type     t_fichier = file of ...;
   var      resultat : t_fichier;      (* deux fichiers internes *)
            copie : t_fichier;
...
```

Compléments : fichiers *input / output*

Il existe deux fichiers de texte prédéfinis : *input* et *output* .

En fait ces deux fichiers représentent :

- ✓ ***le clavier*** (*input*, considéré comme ouvert en lecture)
- ✓ ***l'écran*** (*output* , considéré comme ouvert en écriture)

On a jusqu'à présent utilisé (sans le savoir) les abréviations suivantes:

```
read ( v1, v2 ..., vn );  pour read ( input, v1, v2, ..., vn );
write ( e1, e2, ..., en ); pour write ( output, e1, e2, ..., en );
```

Compléments : directives de compilation

Gestion des erreurs par les directives de compilation

La directive de compilation `I` permet de gérer les erreurs d'entrée-sortie :

- par défaut, elle est active et toutes les opérations d'entrée-sortie sont vérifiées par le système.Toute erreur provoque l'arrêt du programme et un message d'erreur est affiché.
- on peut la désactiver (ponctuellement) avec la directive : `{ $I-} ;` dans ce cas, aucune vérification n'est effectuée à l'exécution
  - ✓il est alors de la responsabilité du programmeur de vérifier et corriger les erreurs
  - ✓grâce à la fonction standard `IOresult`, il est possible de savoir si l'opération d'entrée-sortie s'est bien passée (retourne 0)

Exemple : le programme demande le nom du fichier jusqu'à ce qu'un nom de fichier existant soit spécifié :

```
repeat
  write('Donnez le nom physique du fichier ?');readln(NomPhysique);
  assign(BU,NomPhysique+'.DAT');
  { $I-} reset(BU);{ $I+}
  OK:=(IOresult=0);
  if not OK then
    writeln('impossible de trouver le fichier ',NomPhysique);
until OK;
```

Compléments : commandes additionnelles

- **Commandes supplémentaires pour les fichiers**

```
filepos ( < File > );     (* position actuelle du repère *)
filesize ( < File > );    (* taille du fichier en enregistrements *)
```

Rq : ces commandes sont basées sur les enregistrements d'un fichier mais elles peuvent être utilisées avec un fichier texte (moins utile)

```
erase ( < File > );       (* efface le fichier File *)
rename ( < File >, < nouveau_nom > );   (* renomme le fichier *)
```

*Rq : ces commandes **nécessitent que le fichier soit fermé** avant de pouvoir être utilisées*

```
seek ( < File >, < Num > );   (* positionne le repère sur
                                 l'enregistrement de numéro Num *)
```

Rq : la commande `seek` *permet donc d'accéder directement à un enregistrement spécifique, cela peut être très utile dans un gros fichier binaire,*
elle permet d'utiliser un accès direct, et ce même dans un fichier séquentiel, comme nous allons le voir par la suite.

Compléments : accès direct

Accès direct à un enregistrement dans un fichier par la commande `seek`

(cf déclaration des types, page 14)

```
(* déclarations de types *)
(* déclarations des variables *)
var       BU : t_fichier_biblio;
          Courant : t_livre;
          Num : integer;
begin
  (* assignation *)
  (* ouverture *)
      reset (BU);
  (* saisie du numéro de l'enregistrement recherché *)
      repeat
          writeln ('quel est le numéro de l'enregistrement ? ');
          readln (Num);
      until Num in [ 0..filesize(BU)-1];
  (* lecture de l'enregistrement *)
      seek (BU, Num);
      read (BU, Courant);
  (* affichage *)
      with Courant do
      writeln (titre, '/', auteur, '/', annee, '/', ISBN);
  (* fermeture *)
      close (BU);
end.
```

Compléments : recherche dichotomique

Principe de l'algorithme

• l'idée de base est de choisir un enregistrement **e** au hasard et de le comparer au critère de recherche
  - ✓ s'il est égal, on a trouvé ;
  - ✓ s'il est inférieur, l'enregistrement cherché se situe après ;
  - ✓ s'il est supérieur, l'enregistrement cherché se situe avant.

• puis on réitère avec le sous-ensemble d'enregistrements restant, la solution optimale consistant à choisir **e** au milieu de ce sous-ensemble

Rq : si n est le nombre d'enregistrements du fichier, le nombre maximum d'itérations est log2 n

Exercice :
Rechercher le livre dont l'auteur est Irving par dichotomie dans un fichier de livres.

exemple de programmation

• la fonction RechDicho renvoie le numéro de l'enregistrement vérifiant le critère de recherche, ou -1 si aucun enregistrement ne vérifie le critère
• G et D sont les indices courants (gauche et droite) de la recherche dichotomique dans le fichier.

Compléments : recherche dichotomique (2)

Programme de recherche dichotomique

```
function RechDicho (var f: TypeFichier; v: t_auteur_livre) : integer;

var       G, D, m : integer;
          trouve  : boolean;
          courant : t_livre;
Begin

    G := 0; D := filesize (f) - 1;      (* bornes de l'ensemble considéré *)
    trouve := false;

    while (G <= D and not trouve) do
    begin
        m := (G + D) div 2;             (* calcul de la valeur moyenne *)
        seek (f, m);                    (* positionnement du repère de fichier *)
        read (f, courant);              (* lecture de l'enregistrement *)
        if courant.auteur = v then trouve := true (* condition de sortie *)
        else
            if courant.auteur > v then D := m-1   (* trop à droite *)
            else G := m+1;                        (* trop à gauche *)
    end; (* while *)

    if trouve then RechDicho := m       (* détermine l'état de sortie du prg *)
    else RechDicho := -1;

end; (* RechDico *)
```

Compléments : Indexation (1), principe

Hypothèse

On dispose d'un fichier organisé en accès direct.

Principe

Les enregistrements sont indexés dans une table de taille réduite, appelée index, et contenant les clés et les numéros d'enregistrements associés.

Rq : on rappelle que la clé permet d'identifier de façon unique les enregistrements

Exemple

le numéro ISBN d'un livre peut servir de clé
hypothèse : on suppose (pour simplifier) que la clé est constituée par un attribut de la structure

- l'index est **trié** sur la clé
- l'accès à un enregistrement se fait en deux temps :
  - ✓ accès à l'index qui, à partir de la clé, indique le numéro de l'enregistrement correspondant
  - ✓ accès à l'enregistrement

Compléments : Indexation (2), déclaration

Déclarations d'un index de fichier pour une bibliothèque

```
const     long_max_nom = 30; (* nombre maximal de caractères d'un nom *)
          long_max_titre = 80; (* nombre maximal de caractères d'un titre *)
          nb_max_index = 10000; (* nombre maximal de livres *)

type      t_auteur_livre = string [ long_max_nom ];
          t_titre_livre = string [ long_max_titre ];
          t_livre = record          (* représente une fiche bibliographique *)
            titre : t_titre_libre;       (* titre du livre *)
            auteur : t_auteur_livre; (* nom de l'auteur *)
            ISBN : integer;              (* cote en bibliothèque *)
            annee : integer;             (* année de parution *)
          end;
          t_fichier_biblio = file of t_livre;
          t_cle = ... (* dépend du type de l'attribut clé *)
          t_element_index = record
            cle : t_cle;
            numero_element : integer;
          end;


var       BU : t_fichier_biblio;   (* fichier à copier *)
          Table_Index : array[ 1..nb_max_index] of t_element_index;
```

Compléments : Indexation (3), accès

Constitution de l'index

On peut gérer l'index de deux façons :

- l'index peut être ***généré au début du programme*** à partir du fichier de données
- l'index peut être ***stocké lui-même dans un fichier*** et être copié dans le tableau TableIndex au début du programme

Rq : il faudra dans ce cas recopier l'index dans le fichier à la fin du programme s 'il a été modifié.

Recherche associative (recherche d'un élément dans le fichier)

- on recherche l'enregistrement tel que la clé = critère de recherche
- *principe* : la recherche est faite dans l'index ; si un élément de l'index satisfait la condition, on accède à l'enregistrement correspondant dans le fichier de données
- la *recherche dans l'index* peut se faire suivant différentes stratégies, on peut par exemple utiliser la recherche dichotomique.
- *intérêt* de l'indexation : économie

Compléments : Indexation (4), mise à jour

Difficultés de la mise à jour :

ajout d'un enregistrement
l'ajout d'un enregistrement consiste :

- à ajouter l'enregistrement dans le fichier de données (par exemple à la fin)
- à ajouter l'élément correspondant dans l'index (qui doit rester trié)

suppression d'un enregistrement
On distingue :

- la suppression logique de l'enregistrement qui consiste à supprimer l'élément correspondant dans l'index, sans modifier le fichier de données
- la suppression physique qui consiste à supprimer effectivement l'enregistrement du fichier de données

En général, on effectue une *suppression logique dans le cours du traitement* (chaque fois qu'il faut supprimer un enregistrement) et *une seule suppression physique à la fin du traitement*, juste avant la fermeture du fichier.

Conclusion

Intérêt des fichiers : sauvegarder les données.

Manipulation des fichiers en Pascal : fichiers binaires, fichiers textes, méthodes d 'accès.

Deuxième partie du cours : les pointeurs.

Mise en pratique en TD …