
Chapitre 6 FICHIERS



A la fin de cette partie, vous serez capable de:

- Utiliser à bon escient les différents types de chaînes de caractères;
- Utiliser les différentes routines de traitement des chaînes de caractères;

6.1 Différentes sortes de fichiers

Dans les chapitres précédents nous avons introduit des types de données divers, mais dont la particularité était de ne pas trop dépendre du support physique, si ce n'est pour la précision des nombres ou pour la place mémoire nécessaire aux variables que l'on peut déclarer avec de tels types. Afin de pallier la limite imposée par la taille mémoire centrale, mais aussi et surtout de permettre le stockage des informations produites ou modifiées par le programme, le moyen le plus adéquat est l'utilisation des fichiers.

Un fichier est une collection d'informations, enregistrées sur un support physique de mémoire secondaire (mémoire de masse). Il est défini par un identificateur et des attributs tels que le type, les conditions et les méthodes d'accès. Les fichiers sont, de plus, gérés par le système d'exploitation et accessibles par l'intermédiaire de celui-ci.

Delphi dispose de plusieurs possibilités d'enregistrement de données dans des fichiers:

- Les fichiers dans leur forme traditionnelle
- Les fichiers liés et supportés par la VCL, offerts par différentes classes (Tstream, Tcomponent,...) ou supportés par des méthodes de plusieurs composants
- Les fichiers inhérents à l'utilisation de bases de données

Dans ce chapitre nous nous attacherons essentiellement à l'étude des fichiers de la première catégorie. Parmi ceux-ci nous nous arrêterons sur:

- Les fichiers de type texte (ou fichiers ASCII)
- Les fichiers à accès direct (ou fichiers typés)
- Les fichiers non typés

Avant d'examiner les caractéristiques de ces différents types de fichiers, voyons ce qu'ils ont en commun. La notion de fichier recouvre deux objets distincts: un identificateur (nom interne) défini dans un programme et le fichier physique géré par le système d'exploitation (associé à un nom externe). Afin qu'un programme puisse gérer un fichier, ces deux noms doivent être mis en relation par la procédure `AssignFile (NomInterne, NomExterne)`. Lorsque l'utilisation du fichier est terminée, il doit être fermé à l'aide de la procédure `CloseFile (NomInterne)`. Entre ces deux opérations, diverses procédures et fonctions prédéfinies relatives aux fichiers peuvent être utilisées.

La gestion de fichiers, autrement dit la gestion des informations qu'ils contiennent, est un domaine important de la programmation. Toutefois, Delphi met à disposition du programmeur des outils permettant parfois de masquer l'utilisation explicite des fichiers.

6.2 Fichiers de type texte

Ces fichiers, appelés aussi "fichiers ASCII", sont constitués par une suite de caractères groupés en lignes, comme dans un texte. Chaque ligne est terminée par la séquence de caractères CR et LF (Carriage Return et Line Feed), correspondant à un passage au début de la ligne suivante. L'accès à ce type de fichiers peut s'effectuer uniquement de manière séquentielle, c'est-à-dire un élément après l'autre, en partant du premier. Leur contenu peut être visualisé en les chargeant dans un éditeur de texte, par exemple le blocnote (Notepad) de Windows.

Nous allons passer en revue et illustrer quelques-unes des instructions relatives aux fichiers de type texte.

Déclaration de fichier

Le mot réservé `TextFile` est utilisé pour déclarer des fichiers de type texte:

```
var fichier_texte : TextFile;
```

Assignment de fichier

La procédure `AssignFile` permet de relier la variable constituant le nom interne du fichier au fichier proprement dit, spécifié par le chemin permettant de le retrouver sur un disque:

```
AssignFile (fichier_texte, 'C:\AUTOEXEC.BAT');
```

Ouverture de fichier

Après avoir été assigné, un fichier doit encore être ouvert. Cette opération est réalisée par l'une des deux procédures prédéfinies:

```
rewrite (fichier_texte);  
reset (fichier_texte);
```

La procédure `rewrite` ouvre un nouveau fichier en vue d'une opération d'écriture. Si un fichier du même nom existe, il sera détruit et son contenu perdu. La procédure `reset` ouvre un fichier existant en vue d'une opération de lecture ou d'écriture. Lors de l'appel à cette procédure, une erreur d'entrée-sortie est signalée si le fichier spécifié est absent.

Ecriture d'informations

Les deux procédures permettant d'enregistrer des données dans un fichier sont `write` et `writeln`. Le premier paramètre qui leur est transmis est l'identificateur du fichier:

```
write (fichier_texte, 'Voic un début de ligne ');  
writeln (fichier_texte, ' puis on saute à la ligne suivante.');
```

Lecture d'informations

De manière analogue, les procédures `read` et `readln` sont utilisées pour la lecture des données placées dans un fichier de type texte. L'exemple qui suit, illustre la lecture du contenu d'un fichier et son affichage; ligne par ligne dans un composant de type Memo. De plus, le nom du fichier ainsi que le nombre de lignes sont également affichés.



Voici comment se présente le fiche de ce projet. Elle comporte:

- Un Label permettant l'affichage du nom du fichier
- Un Memo dans lequel le contenu du fichier est placé
- Un Label indiquant le nombre de lignes lues
- Un bouton qui déclenche le choix du fichier et l'affichage de son contenu
- Un OpenFileDialog (dialogue d'ouverture de fichier) permettant de choisir le fichier à lire



Voici enfin comment est écrit le programme:

```
unit uftxt1;

interface

uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms,
    Dialogs, StdCtrls;

type
    TFtexte = class(TForm)
        Memo: TMemo;
        nomf: TLabel;
        OD: TOpenDialog;
        lignes: TLabel;
        Lire: TButton;
        procedure LireClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var Ftexte: TFtexte;

implementation

{$R *.DFM}

procedure TFtexte.LireClick(Sender: TObject);
var f      : TextFile;      { fichier texte }
    ligne  : string;       { ligne lue }
    nbl    : integer;      { compteur du nombre de lignes }
begin
    nbl := 0;
    if OD.execute then begin
        nomf.caption := OD.FileName;
        AssignFile (f, OD.FileName);
        Reset (f);
        while not eof (f) do begin
            readln (f, ligne);
            Memo.Lines.Add (ligne);
            inc (nbl);
        end;
        CloseFile (f);
        lignes.caption := inttostr (nbl);
    end;
end;

end.
```

Le cœur du programme est constitué par les lignes suivantes:

```
AssignFile (f, OD.FileName);
Reset (f);
while not eof (f) do begin
    readln (f, ligne);
    Memo.Lines.Add (ligne);
    inc (nbl);
end;
CloseFile (f);
```

Fin de ligne

La fonction `Eoln` retourne une valeur booléenne indiquant si la fin d'une ligne (End Of Line) est atteinte. Dans le fragment de programme suivant, la lecture d'un fichier texte s'effectue caractère par caractère à l'aide de la procédure `read` et, à chaque détection de la fin d'une ligne, un bip est émis:

```
AssignFile (f, OD.FileName);
Reset (f);
while not eof (f) do begin
  read (f, caractere);      { on lit un caractère }
  if eoln (f) then
    MessageBeep (0);       { bip standard }
end;
CloseFile (f);
```

Fin de fichier

La fonction `Eof` retourne une valeur booléenne indiquant si la fin du fichier (End Of File) est atteinte.

Fermeture de fichier

Il est fortement conseillé de fermer tous les fichiers ouverts, à l'aide de la procédure `CloseFile`, avant la fin de l'exécution d'un programme:

```
CloseFile (fichier_texte);
```


6.3 Fichiers à accès direct

Ces fichiers sont structurés en éléments ou enregistrements (record) dont le type peut être quelconque. Les éléments d'un fichier à accès direct se suivent logiquement, mais pas forcément physiquement; ils ont tous la même taille. Le terme accès direct signifie qu'il est possible d'accéder directement à n'importe quel élément, pour autant que l'on spécifie son rang. Voici un exemple de déclarations liées au traitement d'un fichier à accès direct:

```
type element = record
  nom : string[12];
  age : integer;
end;
var  personne : element;
     f       : file of element;
```

Le schéma suivant illustre l'organisation logique des premiers enregistrements du fichier f:

rang	0	1	2	3
	Durant	Mercier	César	Alex
	24	44	37	67



La flèche symbolise le pointeur de fichier.

Chaque enregistrement d'un fichier à accès direct peut être accédé en spécifiant son rang. Un pointeur de fichier mémorise le rang de l'enregistrement concerné par la prochaine opération de lecture ou d'écriture. Lorsqu'un fichier est ouvert, son pointeur indique l'enregistrement de rang zéro, c'est-à-dire le premier enregistrement. Après chaque opération de lecture ou d'écriture, le pointeur de fichier est incrémenté du nombre d'enregistrements lus ou écrits. Un enregistrement est l'unité minimale de transfert d'information entre un fichier et un programme.

Le contenu des fichiers à accès direct est stocké sous forme binaire compactée et n'est donc pas directement affichable à l'écran. Il est également important de savoir que l'on peut accéder de manière séquentielle à un fichier à accès direct.

Très souvent les éléments d'un fichier à accès direct sont de type enregistrement.

Déclaration du type de fichier

La déclaration d'un fichier à accès direct est effectuée à l'aide des mots réservés `file of`:

```
type client = record
    code           : string[10];
    nom            : string[40];
    solde          : real;
    premiere_facture : integer;
    derniere_facture : integer;
end;

var fichier_client : file of client;
    un_client      : client;
    ch             : char;
```

En plus d'une variable de type fichier, il faut définir une variable dont le type est celui des éléments de ce fichier. Cette dernière (`un_client`) est utilisée comme paramètre par les procédures de lecture et d'écriture. Elle tient lieu de variable intermédiaire, ou de mémoire tampon.

Assignment de fichier

La procédure `AssignFile` remplit le même rôle que pour les fichiers texte:

```
assign (fichier_client, 'CLIENT.DAT');
```

Ouverture de fichier

Cette opération est effectuée par l'une des procédures `reset` ou `rewrite`, comme pour les fichiers de type texte:

```
rewrite (fichier_client);      { le fichier peut exister }
reset (fichier_stock);         { le fichier doit exister }
```

L'emploi de la procédure `reset` nécessite l'existence préalable du fichier. En revanche l'utilisation de la procédure `rewrite` crée et ouvre inconditionnellement un fichier. Si un fichier portant le même nom existait déjà, son contenu serait perdu. Il convient donc d'être prudent lors de la création d'un fichier à l'aide de cette procédure.

Positionnement à l'intérieur d'un fichier

A chaque fichier utilisé dans un programme Delphi est associé un pointeur permettant au système et à l'utilisateur de connaître l'emplacement du prochain élément accessible. La procédure `seek (nom_de_fichier, no)` permet de positionner le pointeur associé au fichier `nom_de_fichier` sur l'enregistrement numéro `no`, en vue d'une opération de lecture ou d'écriture. Il faut se rappeler que le premier enregistrement d'un fichier porte le numéro zéro.

Instructions de lecture et d'écriture

La procédure de lecture est `read (nom_de_fichier, liste_elements)` et celle d'écriture est `write (nom_de_fichier, liste_elements)`, où `liste_elements` est une liste de variables du même type que les éléments du fichier. Il est important de savoir qu'à chaque écriture et à chaque lecture le pointeur associé au fichier est automatiquement incrémenté de manière à être positionné sur l'enregistrement suivant. Partant de ce principe, la modification du contenu de l'enregistrement numéro 5 d'un fichier implique, par exemple, les opérations suivantes:

```
seek (fichier_client, 5);
read (fichier_client, un_client);
... { modification des données contenues dans un_client }
seek (fichier_client, 5);
write (fichier_client, un_client);
```

Enregistrement courant et taille d'un fichier

La fonction `FilePos (nom_de_fichier)` fournit le rang de l'enregistrement courant. La fonction `FileSize (nom_de_fichier)` donne la taille du fichier, exprimée en nombre d'éléments. Le fragment de programme suivant permet de déterminer si un fichier est vide:

```
if filesize (f) = 0 then
  writeln ('Fichier vide');
```

Changement de nom et effacement d'un fichier

Ces opérations sont effectuées, respectivement, par les procédures `Rename` et `Erase`. La procédure `Erase` permet d'éliminer un fichier du répertoire dans lequel il figure:

```
rename (fichier_client, 'CLIENT.BAK');
erase (fichier_client);
```

Fin de fichier

La fonction booléenne `Eof (nom_de_fichier)` indique si la fin du fichier `nom_de_fichier` est atteinte, comme pour les fichiers texte

Fermeture de fichier

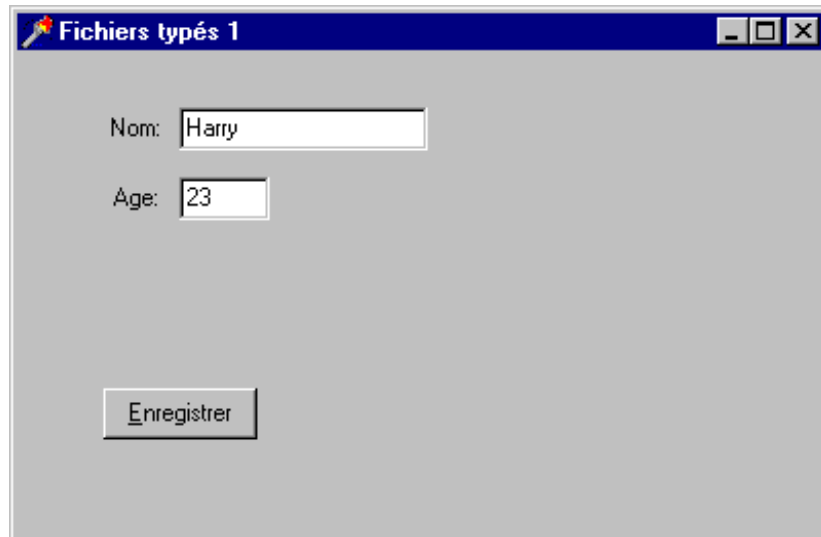
La fermeture d'un fichier à accès direct s'effectue de la même manière que pour un fichier de type texte:

```
CloseFile (fichier_client);
```

1er exemple

Afin d'illustrer l'utilisation d'un fichier à accès direct, examinons ce premier programme permettant d'enregistrer des données (nom et âge de diverses personnes) dans un fichier.

La fenêtre de l'application se présente sous la forme suivante:



Le principe de fonctionnement est le suivant: chaque clic sur le bouton Enregistrer écrit un nouvel enregistrement dans le fichier de données. La première fois (et seulement la première) que l'utilisateur clique sur ce bouton, une boîte de dialogue permettant de choisir le nom du fichier apparaît à l'écran.

Voici maintenant le code du programme:

```
unit udirect1;

interface

uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms,
    Dialogs, StdCtrls;

type
    TPersonne = record
        nom : string[20];
        age : integer;
    end;

    TFichier = file of TPersonne;

    TFtype1 = class(TForm)
        Nom: TEdit;
        Label1: TLabel;
        age: TEdit;
        Label2: TLabel;
        enregistrer: TButton;
        SD: TSaveDialog;
        procedure enregistrerClick(Sender: TObject);
        procedure FormActivate(Sender: TObject);
    end;

end.
```

```

    procedure FormClose(Sender: TObject; var Action:
TCloseAction);
    private
    { Private declarations }
    public
    { Public declarations }
    f      : TFichier;
    pers   : TPersonne;
    nomf   : string;    { nom du fichier }
    end;

var Ftype1: TFtype1;

implementation

{$R *.DFM}

procedure TFtype1.enregistrerClick(Sender: TObject);
begin
    if nomf = '' then          { le fichier n'est pas ouvert }
        if SD.execute then begin
            nomf := SD.filename;
            caption := 'Fichiers typés [' + nomf + ']';
            assignfile (f, nomf);
            rewrite (f);
        end;
    if nomf <> '' then begin
        pers.nom := Nom.text;      { on enregistre les données }
        pers.age := strtoint (Age.text);
        write (f, pers);
        Nom.text := '';
        Age.text := '0';
    end;
end;

procedure TFtype1.FormActivate(Sender: TObject);
begin
    nomf := '';
end;

procedure TFtype1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
    if nomf <> '' then
        closefile (f);
end;

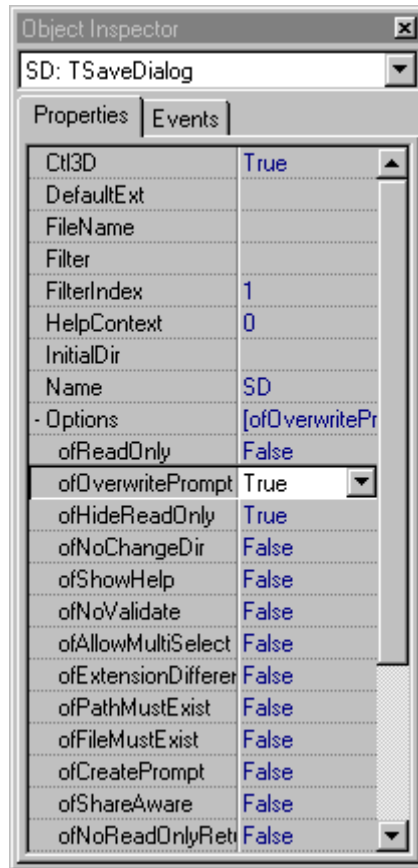
end.

```

En inspectant ce programme on peut noter les points suivants:

- La manière de déclarer les types TPersonne et Tfichier
- L'emplacement où sont déclarées les variables f, pers et nomf
- La variable nomf (contenant le nom du fichier) permet de savoir si le fichier a déjà été ouvert (nomf<>'')
- Le nom du fichier ouvert est placé dans la barre de titre de l'application
- SD est le nom du composant OpenFileDialog
- Parmi les propriétés du composant OpenFileDialog on a spécifié "ofOverwritePrompt=true", c'est-à-dire que le programme met en garde

l'utilisateur lorsque le nom de fichier choisi existe déjà. L'utilisateur peut décider de l'écraser ou de changer le nom choisi.

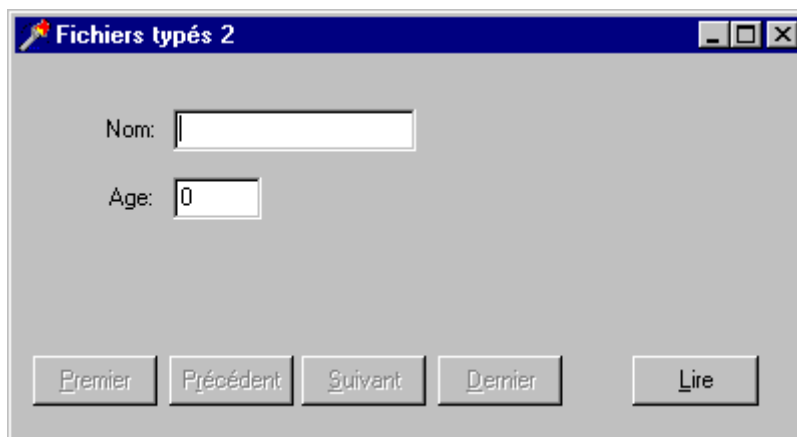


Comme on peut le constater, ce programme ne fait qu'enregistrer des données, les unes à la suite des autres.

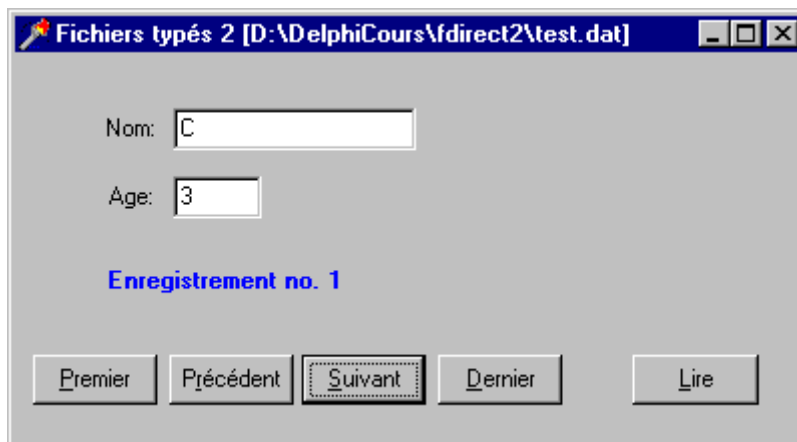
2ème exemple

Le deuxième exemple illustre le déplacement parmi les enregistrements d'un fichier à accès direct ayant la même structure que celui de l'exemple précédent. On pourra donc lire les fichiers créés par ce dernier.

Tant que l'utilisateur n'a pas cliqué sur le bouton Lire, les boutons de déplacement sont désactivés. Le bouton Lire permet justement de choisir le fichier de données à consulter.



Voici comment se présente cette fenêtre après le choix du nom du fichier de données:



L'utilisateur peut alors se déplacer dans le fichier à l'aide des boutons; le numéro de l'enregistrement courant est affiché dans un but de vérification.

Voici le code de ce programme :

```
unit udirect1;

interface

uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms,
    Dialogs, StdCtrls, ExtCtrls, DBCtrls;

type
    TPersonne = record
        nom : string[20];
        age : integer;
    end;

    TFichier = file of TPersonne;

    TFtype1 = class(TForm)
        Nom: TEdit;
        Label1: TLabel;
        age: TEdit;
        Label2: TLabel;
        Lire: TButton;
        OD: TOpenDialog;
        premier: TButton;
        Precedent: TButton;
        Suivant: TButton;
        Dernier: TButton;
        rec: TLabel;
        procedure LireClick(Sender: TObject);
        procedure FormActivate(Sender: TObject);
        procedure FormClose(Sender: TObject; var Action:
TCloseAction);
        procedure premierClick(Sender: TObject);
        procedure SuivantClick(Sender: TObject);
        procedure DernierClick(Sender: TObject);
        procedure PrecedentClick(Sender: TObject);
    end;

end.
```

```

private
  { Private declarations }
public
  { Public declarations }
  f      : TFichier;
  pers   : TPersonne;
  nomf   : string;   { nom du fichier }
  position : integer;
  procedure Afficher;
end;

var Ftypel: TFtypel;

implementation

{$R *.DFM}

procedure TFtypel.Afficher;
begin
  Nom.text := pers.nom;
  Age.Text := inttostr (pers.age);
  Rec.caption := 'Enregistrement no. ' + inttostr (position);
end;

procedure TFtypel.LireClick(Sender: TObject);
begin
  if nomf <> '' then begin
    closefile (f);
    nomf := '';
  end;
  if OD.execute then begin
    nomf := OD.filename;
    caption := 'Fichiers typés 2 [' + nomf + ']';
    assignfile (f, nomf);
    reset (f);
    position := 0;
    read (f, pers);
    Afficher;
    premier.enabled := true;
    dernier.enabled := true;
    suivant.enabled := true;
    precedent.enabled := true;
  end else begin
    premier.enabled := false;
    dernier.enabled := false;
    suivant.enabled := false;
    precedent.enabled := false;
  end;
end;

procedure TFtypel.FormActivate(Sender: TObject);
begin
  nomf := '';
end;

procedure TFtypel.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  if nomf <> '' then
    closefile (f);
end;

```

```

procedure TFtype1.premierClick(Sender: TObject);
begin
  seek (f, 0);
  read (f, pers);
  position := 0;
  Afficher;
end;

procedure TFtype1.SuivantClick(Sender: TObject);
begin
  if not eof (f) then begin
    position := filepos (f);
    read (f, pers);
    Afficher;
  end;
end;

procedure TFtype1.DernierClick(Sender: TObject);
begin
  seek (f, filesize(f)-1);
  position := filepos (f);
  read (f, pers);
  Afficher;
end;

procedure TFtype1.PrecedentClick(Sender: TObject);
begin
  if filepos(f) > 1 then begin
    seek (f, filepos(f)-2);
    position := filepos (f);
    read (f, pers);
    Afficher;
  end;
end;

end.

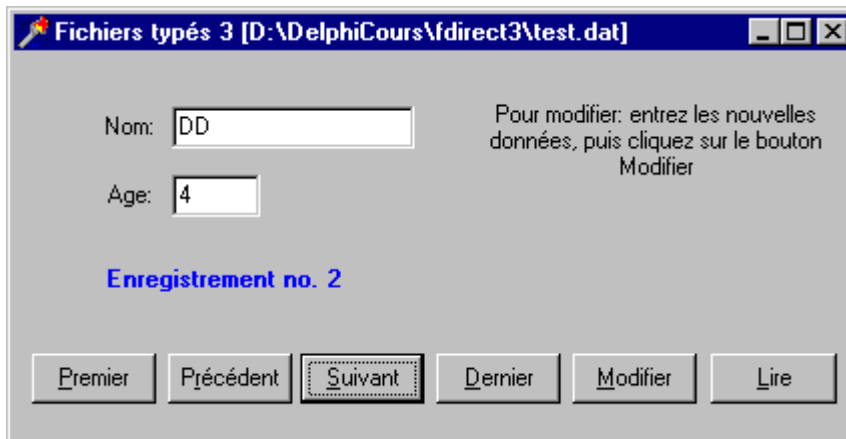
```

En inspectant ce deuxième programme on peut noter les points suivants:

- La variable `position` permet de garder en mémoire la position du pointeur de fichier (numéro de l'enregistrement)
- L'emplacement et la manière de déclarer la procédure `Afficher`
- La variable `nomf` (contenant le nom du fichier) permet de savoir si le fichier a déjà été ouvert (`nomf<>''`)
- La manière dont sont implémentées les quatre opérations de déplacement dans le fichier
- Les sécurités mises en place pour éviter quelques désagréables surprises. Par exemple, ne pas fermer le fichier s'il n'a pas encore été ouvert, ou encore ne pas laisser le programme essayer de se positionner avant le premier enregistrement du fichier
- Après chaque déplacement la procédure `Afficher` est appelée afin de visualiser les données lues,

3ème exemple

Pour terminer nous allons ajouter une fonctionnalité au programme précédent: pouvoir modifier le contenu d'un enregistrement.



Comme on peut le constater, il ressemble beaucoup au programme précédent, sauf le code permettant de modifier un enregistrement, c'est-à-dire la procédure `TFtype1.ModifierClick`.

```
unit udirect1;

interface

uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms,
    Dialogs, StdCtrls, ExtCtrls, DBCtrls;

type
    TPersonne = record
        nom : string[20];
        age : integer;
    end;

    TFichier = file of TPersonne;

    TFtype1 = class(TForm)
        Nom: TEdit;
        Label1: TLabel;
        age: TEdit;
        Label2: TLabel;
        Lire: TButton;
        OD: TOpenDialog;
        premier: TButton;
        Precedent: TButton;
        Suivant: TButton;
        Dernier: TButton;
        rec: TLabel;
        Modifier: TButton;
        Label3: TLabel;
        procedure LireClick(Sender: TObject);
        procedure FormActivate(Sender: TObject);
        procedure FormClose(Sender: TObject; var Action:
TCloseAction);
        procedure premierClick(Sender: TObject);
```

```

    procedure SuivantClick(Sender: TObject);
    procedure DernierClick(Sender: TObject);
    procedure PrecedentClick(Sender: TObject);
    procedure ModifieurClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
    f      : TFichier;
    pers   : TPersonne;
    nomf   : string;    { nom du fichier }
    position : integer;
    procedure Afficher;
end;

var  Ftypel: TTypel;

implementation

{$R *.DFM}

procedure TTypel.Afficher;
begin
    Nom.text := pers.nom;
    Age.Text := inttostr (pers.age);
    Rec.caption := 'Enregistrement no. ' + inttostr (position);
end;

procedure TTypel.LireClick(Sender: TObject);
var ODres : boolean; { variable auxiliaire }
begin
    if nomf <> '' then begin
        closefile (f);
        nomf := '';
    end;
    ODres := OD.execute;
    if ODres then begin
        nomf := OD.filename;
        caption := 'Fichiers typés 3 [' + nomf + ']';
        assignfile (f, nomf);
        reset (f);
        position := 0;
        read (f, pers);
        Afficher;
    end;
    premier.enabled := ODres;
    dernier.enabled := ODres;
    suivant.enabled := ODres;
    precedent.enabled := ODres;
    modifieur.Enabled := ODres;
end;

procedure TTypel.FormActivate(Sender: TObject);
begin
    nomf := '';
end;

procedure TTypel.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
    if nomf <> '' then

```

```

        closefile (f);
end;

procedure TFtype1.premierClick(Sender: TObject);
begin
    seek (f, 0);
    read (f, pers);
    position := 0;
    Afficher;
end;

procedure TFtype1.SuivantClick(Sender: TObject);
begin
    if not eof (f) then begin
        position := filepos (f);
        read (f, pers);
        Afficher;
    end;
end;

procedure TFtype1.DernierClick(Sender: TObject);
begin
    seek (f, filesize(f)-1);
    position := filepos (f);
    read (f, pers);
    Afficher;
end;

procedure TFtype1.PrecedentClick(Sender: TObject);
begin
    if filepos(f) > 1 then begin
        seek (f, filepos(f)-2);
        position := filepos (f);
        read (f, pers);
        Afficher;
    end;
end;

procedure TFtype1.ModifierClick(Sender: TObject);
begin
    pers.nom := Nom.text;          { on stocke les nouvelles données }
    pers.age := strtoint (Age.text);
    seek (f, position);           { on se replace à la position
courante }
    write (f, pers);              { on écrit les données }
    seek (f, position);           { on se replace à la position
courante }
end;

end.

```

Nous pourrions multiplier les exemples, de gestions de stock en facturations, mais cela ne paraît pas souhaitable étant donné que Delphi excelle également dans la gestion des bases de données. Dès que la complexité d'une application gérant des fichiers augmente, il est plus aisé de la développer en faisant appel aux outils spécifiques aux bases de données

6.4 Fichiers non typés

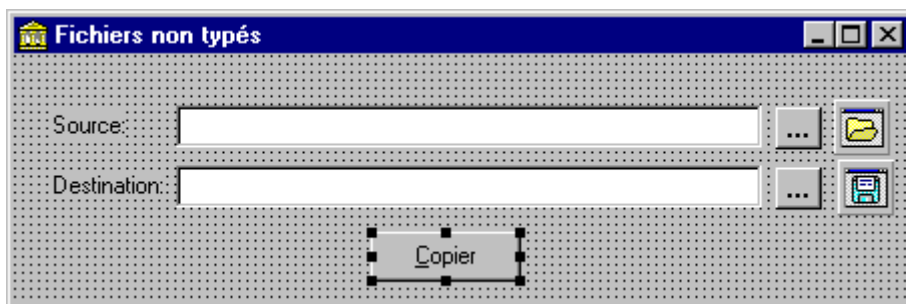
Les fichiers non typés se déclarent à l'aide du mot réservé `file`:

```
var fichier: file;      { et non: file of... }
```

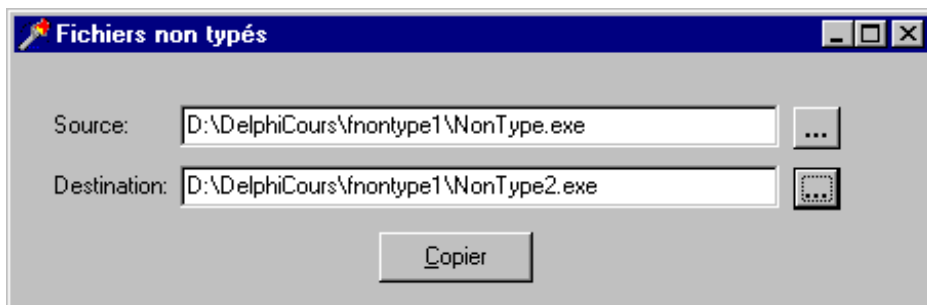
Une variable de type `file` permet d'accéder à un fichier quelconque, sans connaître sa structure ni son contenu. Les opérations de lecture et d'écriture sont effectuées par les procédures `BlockRead` et `BlockWrite`. Les opérations d'assignation et de fermeture sont les mêmes que pour les fichiers à accès direct ou de type texte.

Voici un exemple permettant de copier un fichier, quel que soit son type. L'utilisateur choisit un nom de fichier source, un nom de fichier destination, puis lance la copie.

Voici comment se présente la fiche en version "construction":



Et en version "exécution":



Le code de cette application est le suivant:

```
unit UfnonType;
interface

uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs,
    StdCtrls;

type
    TFnonType = class(TForm)
        OD: TOpenDialog;
        SD: TSaveDialog;
        source: TEdit;
        Label1: TLabel;
    end;
end;
```

```

    dest: TEdit;
    Label2: TLabel;
    ChoixSource: TButton;
    ChoixDest: TButton;
    copie: TButton;
    procedure ChoixSourceClick(Sender: TObject);
    procedure ChoixDestClick(Sender: TObject);
    procedure copieClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var FnonType: TFnonType;

implementation

{$R *.DFM}

procedure TFnonType.ChoixSourceClick(Sender: TObject);
begin
    if OD.execute then
        source.text := OD.filename;
end;

procedure TFnonType.ChoixDestClick(Sender: TObject);
begin
    if SD.execute then
        dest.text := SD.FileName;
end;

procedure TFnonType.copieClick(Sender: TObject);
var src, desti: file;           { les 2 fichiers }
    Nblu, Nbecrit: Integer;     { nbre de bytes lus et écrits }
    Buf: array[1..2048] of Char; { buffer de transfert }
begin
    if not FileExists (source.text) or not FileExists (dest.text)
    then
        begin
            Messagebeep (0);
            exit;
        end;
    AssignFile (src, source.text);
    Reset (src, 1);              { taille d'un record = 1 }
    AssignFile (desti, dest.text);
    Rewrite (desti, 1);         { taille d'un record = 1 }
    repeat
        BlockRead (src, Buf, SizeOf (Buf), Nblu);
        BlockWrite (desti, Buf, Nblu, Nbecrit);
    until (Nblu = 0) or (Nbecrit <> Nblu);
    CloseFile (src);
    CloseFile (desti);
end;

end.

```

En regardant ce programme on peut noter les points suivants:

- La manière dont la boucle `repeat...until` est écrite, ainsi que l'utilisation de `BlockRead` et `BlockWrite`

- Avant de commencer la copie, plutôt que faire des tests compliqués du contenu des zones de texte concernant les noms des fichiers, on fait appel à la fonction bien pratique `FileExists`

6.5 Détection des erreurs d'entrée-sortie

Normalement, lorsqu'une erreur d'entrée-sortie survient, l'exécution du programme est interrompue. Il est possible d'éviter cette interruption intempestive et de gérer depuis le programme l'erreur qui en est la cause. Pour cela, il faut, avant une opération d'entrée-sortie, désactiver la détection des erreurs à l'aide de la directive de compilation `{$I-}` et, après l'opération, invoquer la fonction `IoResult` afin de connaître l'erreur éventuelle. Cette fonction fournit un nombre entier indiquant un code d'erreur lié à la dernière opération d'entrée-sortie. Si ce code vaut zéro, l'opération s'est déroulée sans problème. A l'aide de la directive de compilation `{$I+}`, le compilateur reprend en charge la détection des erreurs d'entrée-sortie.

Le fragment de programme qui suit montre comment vérifier l'existence d'un fichier dont le nom est fourni par l'utilisateur:

```

...
begin
  repeat
    SaisieNomFichier;
    assignfile (fichier, nom_fichier);
    {$I-}          { traitement des erreurs désactivé }
    reset (fichier);
    io := ioresult;   { io contient le code d'erreur   }
    {$I+}          { traitement des erreurs réactivé  }
    if io <> 0 then
      Res.text := 'ERREUR : ' + inttostr (io);
  until io = 0;      { pas d'erreur, le fichier existe }
end;

```

Après invocation de la fonction `IoResult`, celle-ci retourne la valeur zéro jusqu'à la prochaine opération d'entrée-sortie; c'est pourquoi il faut affecter son résultat à une variable en vue du traitement de l'erreur par le programme. Dans une suite d'instructions comprises entre `{$I-}` et `{$I+}`, la fonction `IoResult` doit systématiquement être appelée après chaque opération d'entrée-sortie, afin d'éviter des résultats imprévisibles.

Voici un autre exemple utilisant les directives de compilation `{$I-}` et `{$I+}`. Il s'agit de l'exemple de la section 6.4, modifié afin d'incorporer ces directives:

```

procedure TFnonType.copieClick(Sender: TObject);
var src, desti: file;           { les 2 fichiers }
    Nblu, Nbecrit: Integer;     { nbre de bytes lus et écrits }
    Buf: array[1..2048] of Char; { buffer de transfert }
begin
  // On utilise pas de test d'existence de fichiers
  // afin de mettre en évidence l'utilisation de la directive {$I-}
  //-----
  // if not FileExists (source.text) or not FileExists (dest.text)
  // then begin
  //   Messagebeep (0);
  //   exit;
  // end;
  //-----
  AssignFile (src, source.text);
  {$I-}
  Reset (src, 1);                { taille d'un record = 1 }
  {$I+}
  if ioresult > 0 then begin

```

```

        messagedlg ('Erreur d''ouverture du fichier source', mtError,
                    [mbOK],0);
        exit;
    end;
    AssignFile (desti, dest.text);
{$I-}
    Rewrite (desti, 1);                { taille d'un record = 1 }
{$I+}
    if ioreult > 0 then begin
        messagedlg ('Erreur d''ouverture du fichier destination',
                    mtError, [mbOK],0);
        CloseFile (src);
        exit;
    end;
    repeat
        BlockRead (src, Buf, SizeOf (Buf), Nblu);
        BlockWrite (desti, Buf, Nblu, Nbecrit);
    until (Nblu = 0) or (Nbecrit <> Nblu);
    CloseFile (src);
    CloseFile (desti);
end;

```

Avec Delphi, il est toutefois préférable de traiter les erreurs d'accès aux fichiers à l'aide des exceptions (voir plus loin). De plus, l'utilisation de dialogues standard d'ouverture et de sauvegarde de fichiers permet généralement d'éviter les erreurs.