Algorithmique & programmation

Chapitre 3 : Fichiers séquentiels

Entrées/Sorties
Fichiers Binaires à Accès Séquentiel

Fichiers et Ada

- En Ada les entrées/sorties sont supportées par des paquetages standards du langage
- Les déclarations de ces paquetages sont écrites entièrement en Ada
- Chaque représentation/mode d'accès utilise un paquetage particulier qui peut être importé en utilisant une clause `with`

Fichiers texte à accès séquentiel

- On a utilisé jusqu'alors le paquetage **`P_Esiut`**, qui n'est pas standard mais spécifique à l'IUT.

- Le paquetage standard d'Ada pour les fichiers texte est **`Text_io`**
  - Le paquetage **`P_Esiut`** est d'ailleurs construit à partir du paquetage **`Text_io`** (*cf poly Ada*)
  - Le paquetage **`Text_io`** est également utilisé pour les E/S textuelles sur des fichiers autres que le clavier et l'écran

Fichiers binaires à accès séquentiel

- On utilise le paquetage **`Sequential_io`**

- Ce paquetage est **générique**, c'est-à-dire applicable à plusieurs types de données

- Toute utilisation de ce paquetage passe par une phase **d'instanciation** précisant le type des éléments que l'on manipule :

```
package P_Montype_Io
        is new Sequential_Io(T_Montype);
```

Fichiers binaire à accès direct

- On utilise le paquetage `Direct_io`
- Ce paquetage est lui aussi **générique** et devra être instancié avant tout usage.

```
package P_Montype_Direct_Io
                is new Direct_Io(T_Montype);
```

- Les éléments d'un tel fichier sont lus et écrits dans n'importe quel ordre en spécifiant le rang de l'élément à lire ou écrire

Fichiers et Exceptions

- Les E/S sont effectuées par des appels ordinaires à des procédures et/ou fonctions définies dans les paquetages cités
- Pour toute erreur dans une opération d'E/S, une **exception** est déclenchée
- Le nom de l'exception varie selon l'erreur (*cf poly Ada*)

Identification d'un fichier

- Toute opération d'E/S doit identifier le fichier sur lequel les traitements sont exécutés
- Un fichier a un nom unique et permanent appelé nom externe comme par exemple :

```
/users/pub/1a/ada/tp8.adb
```

- Le système d'exploitation maintient un catalogue qui associe à chaque fichier : sa localisation physique, ses droits d'accès, sa date de création ...
- Sous Unix, on peut les visualiser par : `ls -l`

`--rw-r--r-`	**Dupont**	**2705**	**Nov 22 2007**	**tp8.adb**
droits	propriétaire	taille	date	nom externe

Nom Interne d'un fichier

- Dans un programme, un fichier est identifié pas un nom interne (ou variable fichier)
- Ce nom interne est déclaré comme une variable du type **`file_type`**
  - type exporté par les paquetages Ada d'E/S
- Le nom interne a une durée de vie qui est celle du programme
- C'est en utilisant un nom interne que l'on manipulera un fichier lors d'opérations d'E/S

Exemple de déclaration

- fichier binaire séquentiel

```
type TR_Personne is record
          Nom : String(1..10) ;
          Age : Integer range 0..120 ;
end record;

-- instanciation du paquetage générique Sequential_io
-- pour des E/S d'éléments du type TR_Personne
package P_Personne_Io is new Sequential_Io(TR_Personne) ;
use P_Personne_Io ;

-- Exemple de déclaration d'un fichier de TR_Personne
Fpersonne : P_Personne_Io.File_Type ;
```


Ouverture d'un fichier

- L'association entre nom externe et nom interne se fait lors de l'ouverture du fichier, par une instruction **`open`** (ou **`create`**) qui :
  - recherche le fichier par son nom externe dans le catalogue du système
  - détermine la localisation physique du fichier
  - détermine les droits d'accès
  - associe l'identificateur interne Ada et le nom externe du système
- Cette opération d'initialisation par l'usager du nom interne est à réaliser **une seule fois** avant de commencer les traitements

Mode d'utilisation d'un fichier

- Lors de l'ouverture d'un fichier, il faut indiquer le **mode d'utilisation** de ce fichier
- Ce mode d'utilisation du fichier est décrit par un paramètre du type **`file_mode`** exporté par tout paquetage d'E/S
- **`file_mode`** est un type énuméré ayant les valeurs :
  - **`in_file`** : lecture seule
  - **`out_file`** : écriture seule
  - **`inout_file`** : lecture et écriture (uniquement avec **`Direct_io`**)

Exemple : ouvertures, création de fichiers

```
create(Fpersonne, Out_File, "fpers.dat");
-- le fichier "fpers.dat" n'existait pas dans
-- le répertoire

open(Fpersonne, In_File,    "fpers.dat");
-- ouverture en lecture seule

open(Fpersonne, Out_File,   "fpers.dat");
-- l'ancien contenu est détruit, écriture seule

open(Fpersonne, Inout_File, "fpers.dat");
-- ouverture en lecture écriture (Direct_io)
```

Fermeture d'un fichier

- La dissociation entre nom interne et nom externe se fait par l'opération **`close`**
- Après avoir fermé un fichier, on ne peut plus l'utiliser à moins de le ré-ouvrir

```
close(Fpersonne)
-- on ne peut plus manipuler Fpersonne
après cette instruction
```

- Si le programmeur oublie de fermer explicitement un fichier, le système s'en chargera à la fin du programme

E/S sur fichiers binaires séquentiels

- Les opérations de manipulation d'un fichier utilisent le nom interne sans jamais plus faire référence au nom de fichier externe

- Se positionner en début du fichier (*relire* ou *récrire* en *algo*)

```
reset(Fpersonne);

reset(Fpersonne, In_File);
-- changement du mode d'ouverture
-- au moment du reset
```

E/S sur fichiers binaires séquentiels

- Lire ou écrire (selon le mode d'utilisation spécifié lors de l'ouverture ou du reset)

```
-- UnePersonne doit être une variable
-- déclarée de type TR_Pers

read(Fpersonne,Unepersonne);
-- équivalent au lire(f,c) de l'algo

write(Fpersonne,Unepersonne);
-- équivalent au écrire(f,c) de l'algo
```


E/S sur fichiers binaires séquentiels

- Prédicat de détection de la fin de fichier

```
if end_of_file(Fpersonne) then
-- équivalent au fdf(f) de l'algo
```

Exemple Complet – .ads

```
-- Importer le module générique Sequential_io
-- La clause use est inutile dans ce cas
with Sequential_Io;

package P_Personne is
  -- Déclarer le type des éléments contenus dans le fichier
  type Tr_Personne is record
        Nom, Prenom : String(1..30);
  end record;

  -- Instancier le paquetage Sequential_Io
  -- Pour manipuler des fichiers de Tr_Personne
  package P_Personne_Io is new Sequential_Io(Tr_Personne);
  use P_Personne_Io;

  -- Exemple de procédure manipulant un fichier de Personnes
  procedure Affiche (F : in out P_Personne_Io.File_Type );
end P_Personne;
```


Exemple Complet – adb

```
with P_Esiut; use P_Esiut;

package body P_Personne is

  procedure Affiche (F : in out P_Personne_Io.File_Type) is
    -- variable pour lire un élément du fichier
    Personne : TR_Personne;
  begin
    reset(F);  -- on se positionne en début de fichier
    while not End_Of_File(F) loop
      -- tant que la fin de fichier n'est pas atteinte…
      -- on lit un nouvel élément dans le fichier
      read(F, Personne);
      Ecrire_ligne(Personne.nom & ' ' & Personne.prenom);
    end loop;
  end Affiche;

end P_Personne;
```

Exemple Complet – procédure principale

```
with P_Personne; use P_Personne;
use P_Personne.P_Personne_Io;
-- P_Personne_Io est considéré comme un paquetage
-- faisant partie du paquetage P_Personne
procedure Test_Personne is
  -- déclaration d'un fichier de personne
  personne : P_Personne_Io.File_Type;
begin
  -- ouverture du fichier en lecture
  -- le fichier fpersonne.dat devra être présent
  -- dans le même répertoire que le programme test_Personne
  Open(Fpersonne, In_File, "fpersonne.dat");
  -- parcours et affichage du contenu du fichier
  Affiche(Fpersonne);
  Close(Fpersonne);  -- fermeture du fichier après usage
end Test_Personne;
```


Conventions

- Préfixer avec le nom du paquetage lorsqu'il y a risque d'ambiguïté
- Dans l'exemple ci-dessus, l'utilisation de la clause
  `use P_Personne.P_Personne_Io`
  permet de ne pas préfixer les objets exportés par ce module
- C'est ce qui permet d'écrire **`Close(Fpers)`** au lieu de **`P_Personne.P_Personne_Io.close(Fpers)`**
- Cette même clause **`use`** pourrait permettre de ne pas préfixer le type **`File_Type`** comme on le fait pourtant :
  `Fpersonne : P_Personne_Io.File_Type;`
- C'est pour la lisibilité du programme que l'on préfixe ici : cela permet de connaître d'un seul coup d'œil le type des éléments contenus dans le fichier **`Fpers`**
- Ce préfixage sera plus **indispensable** si, dans un même programme, vous manipulez deux types de fichiers différents en ayant donc instancié deux paquetages d'E/S différents qui définiront tous les deux un type **`File_Type`**

Conventions

- Faire bien attention à ne pas confondre :
  - le paquetage d'E/S instancié défini par :
    - `package P_Personne_Io is new Sequential_Io(Tr_Personne)`
  - avec la variable fichier, de type **`File_Type`**, défini par ce paquetage :
    - `Fpersonne : P_Personne_Io.File_Type;`
  - ou le nom externe, nom du fichier reconnu par Unix, qui est une chaîne de caractère (de type **`String`**)
    - `"fpersonne.dat"`
- Pour cela, utiliser systématiquement les conventions de nommage suivantes :
  - le type des éléments d'un fichier : **`T_Elem`**
    (avec **`T`** pour type, ou **`TR`** pour type record)
  - le module instancié : **`P_Elem_IO`**
    (avec **`P`** pour package et **`IO`** pour Input/Out)
  - le nom interne (objet fichier) : **`Felem`**
    (avec **`F`** comme fichier)
  - le nom externe (fichier disque) : **`"felem.dat"`**