Parte I: Elementos del lenguaje Ada

UNIVERSIDAD DE CANTABRIA

1. Introducción a los computadores y su programación
2. Elementos básicos del lenguaje
3. Modularidad y programación orientada a objetos
4. Estructuras de datos dinámicas
5. Tratamiento de errores
6. Abstracción de tipos mediante unidades genéricas
7. *Entrada/salida con ficheros*
8. Herencia y polimorfismo
9. Programación concurrente y de tiempo real

7.1. Introducción

UNIVERSIDAD DE CANTABRIA

Hay dos tipos de entrada/salida según el tipo de información:

- de *texto*
  - pensada para los “humanos”
  - restringida a datos que se puedan convertir a texto (strings, caracteres, números, enumerados)
  - basada en *líneas* de caracteres
- *binaria*
  - pensada para las “máquinas”
  - para cualquier tipo de dato, excepto punteros
  - las estructuras con punteros deben ser “serializadas”
    - con funciones especiales de lectura y escritura que recuperan o guardan la información apuntada
  - basadas en secuencias de “casillas” o de bytes

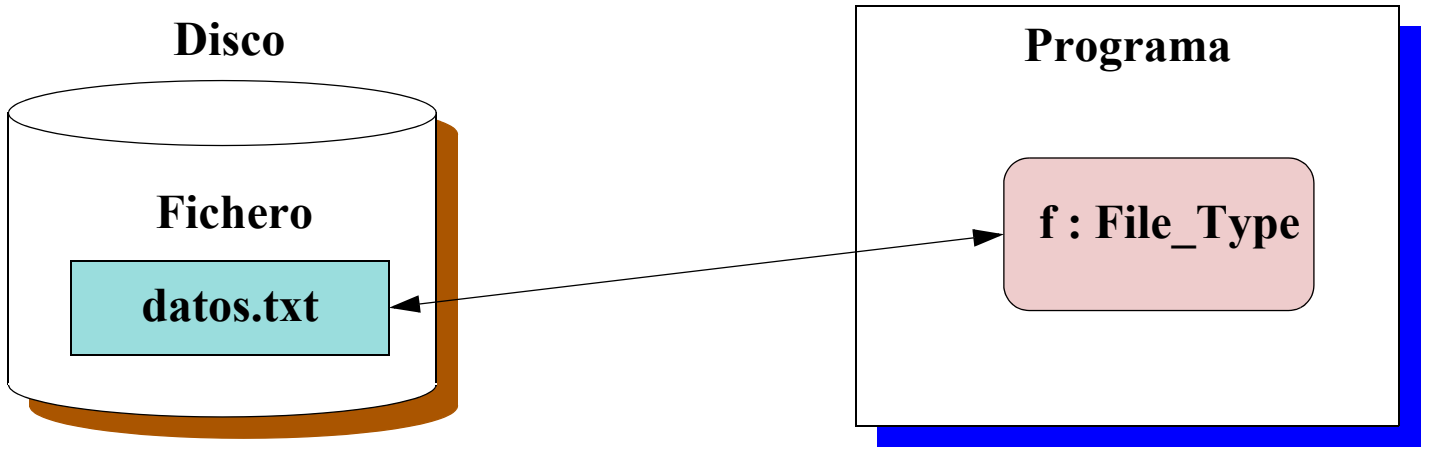
7.2. Ficheros

Representan una estructura de datos almacenada en memoria secundaria o un dispositivo de entrada/salida

- Si es memoria secundaria, los datos son *no volátiles*

Se identifican en el sistema operativo mediante un *nombre*

Se representan en el programa mediante una estructura de datos de un tipo `File_Type`



Uso de ficheros

UNIVERSIDAD DE CANTABRIA

El uso siempre es así:

- *Abrir* el fichero para establecer una asociación entre la estructura de datos y el fichero
- *Leer* y/o *escribir*
- *Cerrar* el fichero para desvincular la asociación y dejar los datos en un estado consistente

Hay unos ficheros de texto ya abiertos, para uso habitual

- entrada estándar (generalmente el teclado)
- salida estándar (generalmente la pantalla)
- salida de error (generalmente la pantalla)

Se pueden modificar haciendo que sean otros ficheros

Interfaz para los ficheros

UNIVERSIDAD DE CANTABRIA

Abrir

- 2 modalidades:
  - `Create`: para crear un fichero nuevo (se borra el viejo)
  - `Open`: para ficheros existentes (si no existe, falla)
- requiere indicar el modo (del tipo `File_Mode`)
  - `In_File` (leer), `Out_File` (escribir), `Append_file` (añadir al final), `Inout_File` (leer y escribir)
- también requiere la variable del tipo `File_Type` y el nombre del fichero (string)

Cerrar: `Close`

Leer y *escribir*: depende del tipo de fichero

7.3. Entrada/salida de texto

UNIVERSIDAD DE CANTABRIA

Está en el paquete `Ada.Text_IO`

Ya la conocemos para entrada/salida estándar

- ej.: operaciones `Get`, `Put`, `Get_Line`, `Put_Line`

Existen versiones de estas operaciones para otros ficheros

- el primer parámetro de estas operaciones es la variable `File_Type`

Notas:

UNIVERSIDAD DE CANTABRIA

El paquete `Ada.Text_IO`, además de la entrada/salida de texto para cualquier fichero, integra varios paquetes genéricos que se pueden instanciar para entrada/salida de los siguientes tipos:

- enteros: `Integer_IO`
- reales: `Float_IO`
- enumerados: `Enumeration_IO`
- enteros modulares: `Modular_IO`
- reales en coma fija: `Fixed_IO`
- reales decimales: `Decimal_IO`

Existen dos instancias muy útiles de estos paquetes para los tipos integer y float:

- `Integer_Text_IO`
- `Float_Text_IO`

Resumen de la especificación de Ada.Text_IO (1/14)

UNIVERSIDAD DE CANTABRIA

```
with Ada.IO_Exceptions;
package Ada.Text_IO is

   type File_Type is limited private;

   type File_Mode is (In_File, Out_File, Append_File);

   type Count is range 0 .. System.Parameters.Count_Max;

   subtype Positive_Count is Count range 1 .. Count'Last;

   subtype Number_Base is Integer range 2 .. 16;

   type Type_Set is (Lower_Case, Upper_Case);
```

Especificación de Ada.Text_IO (2/14)

UNIVERSIDAD DE CANTABRIA

```
-- File Management
procedure Create (File : in out File_Type;
                  Mode : in File_Mode := Out_File;
                  Name : in String := "";
                  Form : in String := "");
procedure Open   (File : in out File_Type;
                  Mode : in File_Mode;
                  Name : in String;
                  Form : in String := "");
procedure Close  (File : in out File_Type);
procedure Delete (File : in out File_Type);
procedure Reset  (File : in out File_Type; Mode : in File_Mode);
procedure Reset  (File : in out File_Type);

function Mode (File : in File_Type) return File_Mode;
function Name (File : in File_Type) return String;

function Is_Open (File : in File_Type) return Boolean;
```

Especificación de Ada.Text_IO (3/14)

UNIVERSIDAD DE CANTABRIA

```
-- Control of default input, output and error files

procedure Set_Input  (File : in File_Type);
procedure Set_Output (File : in File_Type);
procedure Set_Error  (File : in File_Type);

function Standard_Input  return File_Type;
function Standard_Output return File_Type;
function Standard_Error  return File_Type;

function Current_Input  return File_Type;
function Current_Output return File_Type;
function Current_Error  return File_Type;

-- Buffer control
procedure Flush (File : in File_Type);
procedure Flush;
```

Especificación de Ada.Text_IO (4/14)

UNIVERSIDAD DE CANTABRIA

```
-- Column and Line Control
procedure New_Line (File : in File_Type;
                    Spacing : in Positive_Count := 1);
procedure New_Line (Spacing : in Positive_Count := 1);

procedure Skip_Line (File : in File_Type;
                     Spacing : in Positive_Count := 1);
procedure Skip_Line (Spacing : in Positive_Count := 1);

function End_Of_Line (File : in File_Type) return Boolean;
function End_Of_Line return Boolean;

function End_Of_File (File : in File_Type) return Boolean;
function End_Of_File return Boolean;
```

Especificación de Ada.Text_IO (5/14)

UNIVERSIDAD DE CANTABRIA

```
procedure Set_Col (File : in File_Type;
                   To   : in Positive_Count);
procedure Set_Col (To   : in Positive_Count);

procedure Set_Line (File : in File_Type;
                    To   : in Positive_Count);
procedure Set_Line (To   : in Positive_Count);

function Col (File : in File_Type) return Positive_Count;
function Col return Positive_Count;

function Line (File : in File_Type) return Positive_Count;
function Line return Positive_Count;
```

Especificación de Ada.Text_IO (6/14)

UNIVERSIDAD DE CANTABRIA

```
-- Characters Input-Output

procedure Get (File : in File_Type; Item : out Character);
procedure Get (Item : out Character);
procedure Put (File : in File_Type; Item : in Character);
procedure Put (Item : in Character);
procedure Look_Ahead (File        : in File_Type;
                      Item        : out Character;
                      End_Of_Line : out Boolean);
procedure Look_Ahead (Item        : out Character;
                      End_Of_Line : out Boolean);
procedure Get_Immediate (File     : in File_Type;
                         Item     : out Character);
procedure Get_Immediate (Item     : out Character);
procedure Get_Immediate (File     : in File_Type;
                         Item     : out Character;
                         Available: out Boolean);
procedure Get_Immediate (Item     : out Character;
                         Available: out Boolean);
```

Especificación de Ada.Text_IO (7/14)

UNIVERSIDAD DE CANTABRIA

```
-- Strings Input-Output

procedure Get (File : in File_Type; Item : out String);
procedure Get (Item : out String);
procedure Put (File : in File_Type; Item : in String);
procedure Put (Item : in String);

procedure Get_Line (File : in File_Type;
                    Item : out String;
                    Last : out Natural);

procedure Get_Line (Item : out String;
                    Last : out Natural);

procedure Put_Line (File : in File_Type;
                    Item : in String);

procedure Put_Line (Item : in String);
```

Especificación de Ada.Text_IO (8/14)

UNIVERSIDAD DE CANTABRIA

```
generic
   type Num is range <>;

package Integer_IO is

   Default_Width : Field := Num'Width;
   Default_Base  : Number_Base := 10;

   procedure Get (File  : in File_Type;
                  Item  : out Num;
                  Width : in Field := 0);

   procedure Get (Item  : out Num;
                  Width : in Field := 0);

   procedure Put (File  : in File_Type;
                  Item  : in Num;
                  Width : in Field := Default_Width;
                  Base  : in Number_Base := Default_Base);
```

Especificación de Ada.Text_IO (9/14)

UNIVERSIDAD DE CANTABRIA

```
   procedure Put (Item  : in Num;
                  Width : in Field := Default_Width;
                  Base  : in Number_Base := Default_Base);

   procedure Get (From : in String;
                  Item : out Num;
                  Last : out Positive);

   procedure Put (To   : out String;
                  Item : in Num;
                  Base : in Number_Base := Default_Base);

end Integer_IO;
```

Especificación de Ada.Text_IO (10/14)

UNIVERSIDAD DE CANTABRIA

```
generic
   type Num is digits <>;
package Float_IO is
   Default_Fore : Field := 2; --valores fijados por el compilador
   Default_Aft  : Field := Num'Digits - 1;
   Default_Exp  : Field := 3;

   procedure Get (File  : in File_Type;
                  Item  : out Num;
                  Width : in Field := 0);

   procedure Get (Item  : out Num;
                  Width : in Field := 0);

   procedure Put (File : in File_Type;
                  Item : in Num;
                  Fore : in Field := Default_Fore;
                  Aft  : in Field := Default_Aft;
                  Exp  : in Field := Default_Exp);
```

Especificación de Ada.Text_IO (11/14)

UNIVERSIDAD DE CANTABRIA

```
   procedure Put (Item : in Num;
                  Fore : in Field := Default_Fore;
                  Aft  : in Field := Default_Aft;
                  Exp  : in Field := Default_Exp);

   procedure Get (From : in String;
                  Item : out Num;
                  Last : out Positive);

   procedure Put (To   : out String;
                  Item : in Num;
                  Aft  : in Field := Default_Aft;
                  Exp  : in Field := Default_Exp);

end Float_IO;
```

Especificación de Ada.Text_IO (12/14)

UNIVERSIDAD DE CANTABRIA

```
generic
   type Enum is (<>);
package Enumeration_IO is

   Default_Width : Field := 0; -- valor fijado por el compilador
   Default_Setting : Type_Set := Upper_Case;

   procedure Get (File : in File_Type; Item : out Enum);
   procedure Get (Item : out Enum);

   procedure Put (File  : in File_Type;
                  Item  : in Enum;
                  Width : in Field := Default_Width;
                  Set   : in Type_Set := Default_Setting);

   procedure Put (Item  : in Enum;
                  Width : in Field := Default_Width;
                  Set   : in Type_Set := Default_Setting);
```

Especificación de Ada.Text_IO (13/14)

UNIVERSIDAD DE CANTABRIA

```
   procedure Get (From : in String;
                  Item : out Enum;
                  Last : out positive);

   procedure Put (To   : out String;
                  Item : in Enum;
                  Set  : in Type_Set := Default_Setting);

end Enumeration_IO;

-- Además, en Ada.Text_IO están los paquetes:
--    Modular_IO, para números enteros modulares
--    Fixed_IO, para números reales de coma fija
--    Decimal_IO, para números reales decimales
```

Especificación de Ada.Text_IO (14/14)

UNIVERSIDAD DE CANTABRIA

```
   --  Exceptions

   Status_Error : exception renames IO_Exceptions.Status_Error;
   Mode_Error   : exception renames IO_Exceptions.Mode_Error;
   Name_Error   : exception renames IO_Exceptions.Name_Error;
   Use_Error    : exception renames IO_Exceptions.Use_Error;
   Device_Error : exception renames IO_Exceptions.Device_Error;
   End_Error    : exception renames IO_Exceptions.End_Error;
   Data_Error   : exception renames IO_Exceptions.Data_Error;
   Layout_Error : exception renames IO_Exceptions.Layout_Error;

private

   ...

end Ada.Text_IO;
```

Excepciones en la entrada/salida

UNIVERSIDAD DE CANTABRIA

Excepción	Causa
`Status_Error`	Abrir un fichero ya abierto, cerrar uno ya cerrado, usar uno no abierto
`Mode_Error`	Leer si se ha abierto para escribir, escribir si se ha abierto para leer
`Name_Error`	Al hacer `Open`, el fichero no existe
`Use_Error`	Uso del fichero incompatible con las restricciones del sistema operativo (p.e., usar sin permiso)
`Device_Error`	Error en el dispositivo (p.e., disquete malo)
`End_Error`	Intentar leer pasado el final del fichero
`Data_Error`	Formato de los datos incorrecto (p.e, encontrar letras al leer un número)
`Layout_error`	Error con el formato de líneas (líneas demasiado largas)

Ejemplo de entrada/salida de texto

UNIVERSIDAD DE CANTABRIA

Leer y escribir los datos de una lista de rectángulos

Cada rectángulo tiene

- coordenadas de las esquinas
  - inferior izquierda
  - superior derecha
- color (tipo enumerado)

El fichero tiene un rectángulo por línea, con el formato:

```
20 25 30 45 Rojo
50 67 80 120 Verde
...
```

Notas:

UNIVERSIDAD DE CANTABRIA

En este ejemplo se muestra la entrada/salida de texto (`Ada.Text_IO`), de números enteros (`Ada.Integer_Text_IO`) y de enumerados (con una instancia de `Ada.Text_IO.Enumeration_IO`).

El ejemplo tiene dos procedimientos que hacen los siguiente:

- leer de un fichero un número determinado de rectángulos (se pasa como parámetro) y lo devuelve en un array de rectángulos
- escribir en un fichero un número determinado de rectángulos (se pasa como parámetro) que se pasan en un array de rectángulos

Ejemplo de entrada/salida de texto (1/5)

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_Io,Ada.Integer_Text_Io;
use Ada.Text_Io,Ada.Integer_Text_Io;
procedure Pinta_Rectangulos is

   Max : constant Integer:=100;

   type Color is (Rojo, Verde, Azul);
   package Color_Io is new Enumeration_Io(Color);

   type Coordenadas is record
      X,Y : Integer;
   end record;

   type Rectangulo is record
      Pto_Inf_Izqdo,
      Pto_Sup_Dcho : Coordenadas;
      El_Color     : Color;
   end record;

   type Rectangulos is array (1..Max) of Rectangulo;
```

Ejemplo de entrada/salida de texto (2/5)

UNIVERSIDAD DE CANTABRIA

```
procedure Lee_Rectangulos (R   : out Rectangulos;
                           Num : out Integer) is
   Fich : File_Type;
   Nombre : String(1..20);
   N_Nombre : Integer range 0..20;
begin -- de Lee_Rectangulos
   Num:=0;
   Put("Introduce nombre fichero entrada:");
   Get_Line(Nombre,N_Nombre);
   Open(Fich,In_File, Nombre(1..N_Nombre));
   while not End_Of_File(Fich) loop
      Num:=Num+1;
      Get(Fich,R(Num).Pto_Inf_Izqdo.X);
      Get(Fich,R(Num).Pto_Inf_Izqdo.Y);
      Get(Fich,R(Num).Pto_Sup_Dcho.X);
      Get(Fich,R(Num).Pto_Sup_Dcho.Y);
      Color_Io.Get(Fich,R(Num).El_Color);
      Skip_Line(Fich);
   end loop;
   Close(Fich);
```

Ejemplo de entrada/salida de texto (3/5)

UNIVERSIDAD DE CANTABRIA

```
exception
   when Data_Error =>
      Put_Line("Error formato linea "&Integer'Image(Num));
      Num:=Num-1;
   when End_Error =>
      Put_Line("Incompleta linea "&Integer'Image(Num));
      Num:=Num-1;
   when Name_Error =>
      Put_Line("Fichero No Existe ");
end Lee_Rectangulos;


procedure Escribe_Rectangulos
   (R   : in Rectangulos;
    Num : in Integer)
is
    Fich : File_Type;
    Nombre : String(1..20);
    N_Nombre : Integer range 0..20;
```

Ejemplo de entrada/salida de texto (4/5)

UNIVERSIDAD DE CANTABRIA

```
begin
   Put("Introduce nombre fichero salida:");
   Get_Line(Nombre,N_Nombre);
   Create(Fich,Out_File, Nombre(1..N_Nombre));
   for I in 1..Num loop
      Put(Fich,R(I).Pto_Inf_Izqdo.X,8);
      Put(Fich,R(I).Pto_Inf_Izqdo.Y,8);
      Put(Fich,R(I).Pto_Sup_Dcho.X,8);
      Put(Fich,R(I).Pto_Sup_Dcho.Y,8);
      Set_Col(Fich,40);
      Color_Io.Put(Fich,R(I).El_Color);
      New_Line(Fich);
   end loop;
   Close(Fich);
end Escribe_Rectangulos;
```

Ejemplo de entrada/salida de texto (5/5)

UNIVERSIDAD DE CANTABRIA

```
   Los_Rectangulos : Rectangulos;
   Num : Integer range 0..Max:=0;

begin -- programa principal
   Lee_Rectangulos(Los_Rectangulos, Num);
   Escribe_Rectangulos(Los_Rectangulos, Num);
   -- etc.
end Pinta_Rectangulos;
```

7.4. Tipos de entrada/salida binaria

UNIVERSIDAD DE CANTABRIA

3 modalidades para cualquier tipo de datos

- 2 para casillas uniformes (siempre del mismo tipo)
  - entrada/salida *secuencial*
  - entrada/salida *directa*, con acceso aleatorio (similar a un array, pero en disco)
- 1 para casillas de tipos diferentes
  - entrada/salida de *streams*
  - es secuencial
  - es fácil equivocarse, si se leen datos en orden distinto a como se han escrito

7.5. Paquete Ada.Sequential_IO (1/3)

UNIVERSIDAD DE CANTABRIA

```
with Ada.IO_Exceptions;
generic
   type Element_Type (<>) is private;

package Ada.Sequential_IO is

   type File_Type is limited private;

   type File_Mode is (In_File, Out_File, Append_File);

   -- File management
   procedure Create (File : in out File_Type;
                     Mode : in File_Mode := Out_File;
                     Name : in String := "";
                     Form : in String := "");
   procedure Open   (File : in out File_Type;
                     Mode : in File_Mode;
                     Name : in String;
                     Form : in String := "");
```

Paquete Ada.Sequential_IO (2/3)

UNIVERSIDAD DE CANTABRIA

```
procedure Close  (File : in out File_Type);
procedure Delete (File : in out File_Type);
procedure Reset  (File : in out File_Type; Mode : in File_Mode);
procedure Reset  (File : in out File_Type);

function Mode    (File : in File_Type) return File_Mode;
function Name    (File : in File_Type) return String;
function Form    (File : in File_Type) return String;

function Is_Open (File : in File_Type) return Boolean;

-- Input and output operations

procedure Read  (File : in File_Type; Item : out Element_Type);
procedure Write (File : in File_Type; Item : in Element_Type);

function End_Of_File (File : in File_Type) return Boolean;
```

Paquete Ada.Sequential_IO (3/3)

UNIVERSIDAD DE CANTABRIA

```
   -- Exceptions

   Status_Error : exception renames IO_Exceptions.Status_Error;
   Mode_Error   : exception renames IO_Exceptions.Mode_Error;
   Name_Error   : exception renames IO_Exceptions.Name_Error;
   Use_Error    : exception renames IO_Exceptions.Use_Error;
   Device_Error : exception renames IO_Exceptions.Device_Error;
   End_Error    : exception renames IO_Exceptions.End_Error;
   Data_Error   : exception renames IO_Exceptions.Data_Error;

private
   ...
end Ada.Sequential_IO;
```

Notas:

UNIVERSIDAD DE CANTABRIA

A continuación se muestra un ejemplo de entrada/salida secuencial para los dos tipos definidos en el ejemplo del Tema 3 (página 46):

- `Alumno` del paquete `Alumnos`
- `Clase` del paquete `Clases`

Se crean dos procedimientos que se añaden al paquete `Clases` para:

- escribir una clase en un fichero cuyo nombre se pasa como parámetro
- leer una clase de un fichero cuyo nombre se pasa como parámetro

Ambos utilizan instancias del paquete genérico `Ada.Sequential_IO` para el tipo `Alumno`

Ejemplo de entrada/salida secuencial (1/4)

UNIVERSIDAD DE CANTABRIA

```
with Alumnos;

package Clases is

   ...

   procedure Escribe
     (La_Clase   : in Clase;
      En         : in String);
   procedure Lee
     (La_Clase   : out Clase;
      En         : in  String);

private
   ...
end Clases;
```

Ejemplo de entrada/salida secuencial (2/4)

UNIVERSIDAD DE CANTABRIA

```
package body Clases is

   ...

   procedure Escribe (La_Clase   : in Clase;
                      En         : in String)
   is
      package Alumno_Io is new (Alumnos.Alumno);
      Fichero : Alumno_Io.File_Type;
   begin
      Alumno_Io.Create(File => Fichero,
                       Name => En);
      for I in 1..La_Clase.Num loop
         Alumno_Io.Write(Fichero,La_Clase.Alumnos(I));
      end loop;
      Alumno_Io.Close(Fichero);
   end Escribe;
```

Ejemplo de entrada/salida secuencial (3/4)

UNIVERSIDAD DE CANTABRIA

```
   procedure Lee (La_Clase   : out Clase;
                  En         : in String)
   is
      package Alumno_Io is new Ada.Sequential_Io(Alumnos.Alumno);
      Fichero : Alumno_Io.File_Type;
   begin
      La_Clase.Num:=0;
      Alumno_Io.Open(File => Fichero,
                     Mode => Alumno_Io.In_File,
                     Name => En);
      while not Alumno_Io.End_Of_File(Fichero) loop
         La_Clase.Num:=La_Clase.Num+1;
         Alumno_Io.Read(Fichero,La_Clase.Alumnos(La_Clase.Num));
      end loop;
      Alumno_Io.Close(Fichero);
   exception
      when Alumno_Io.Name_Error =>null;
   end Lee;

end Clases;
```

Ejemplo de entrada/salida secuencial (4/4)

UNIVERSIDAD DE CANTABRIA

```
with Menu,Alumnos,Clases;
procedure Lista_Alumnos is

   Tercero_B  : Clases.Clase;
   Eleccion   : Menu.Opcion;
   Alu : Alumnos.Alumno;
   Num : Clases.Num_Alumno;

begin
   Clases.Lee(Tercero_B,"datos.dat");
   loop
      Menu.Pide_Opcion (Eleccion);
      case Eleccion is
         when Menu.Insertar => ...
         when Menu.Mirar => ...
         when Menu.Salir => exit;
      end case;
   end loop;
   Clases.Escribe(Tercero_B,"datos.dat");
end Lista_Alumnos;
```

7.6. Paquete Ada.Direct_IO (1/4)

UNIVERSIDAD DE CANTABRIA

```
with Ada.IO_Exceptions;
generic
   type Element_Type is private;
package Ada.Direct_IO is

   type File_Type is limited private;
   type File_Mode is (In_File, Inout_File, Out_File);
   type Count is new System.Direct_IO.Count;
   subtype Positive_Count is Count range 1 .. Count'Last;

    -- File Management --
   procedure Create (File : in out File_Type;
                     Mode : in File_Mode := Inout_File;
                     Name : in String := "";
                     Form : in String := "");
   procedure Open   (File : in out File_Type;
                     Mode : in File_Mode;
                     Name : in String;
                     Form : in String := "");
```

Paquete Ada.Direct_IO (2/4)

UNIVERSIDAD DE CANTABRIA

```
procedure Close  (File : in out File_Type);
procedure Delete (File : in out File_Type);
procedure Reset  (File : in out File_Type; Mode : in File_Mode);
procedure Reset  (File : in out File_Type);

function Mode (File : in File_Type) return File_Mode;
function Name (File : in File_Type) return String;
function Form (File : in File_Type) return String;

function Is_Open (File : in File_Type) return Boolean;

-- Input and Output Operations

procedure Read (File : in File_Type;
                Item : out Element_Type;
                From : in Positive_Count);

procedure Read (File : in File_Type;
                Item : out Element_Type);
```

Paquete Ada.Direct_IO (3/4)

UNIVERSIDAD DE CANTABRIA

```
procedure Write (File : in File_Type;
                 Item : in Element_Type;
                 To   : in Positive_Count);

procedure Write (File : in File_Type;
                 Item : in Element_Type);

procedure Set_Index (File : in File_Type;
                     To   : in Positive_Count);

function Index (File : in File_Type) return Positive_Count;
function Size  (File : in File_Type) return Count;

function End_Of_File (File : in File_Type) return Boolean;
```

Paquete Ada.Direct_IO (4/4)

UNIVERSIDAD DE CANTABRIA

```
   -- Exceptions

   Status_Error : exception renames IO_Exceptions.Status_Error;
   Mode_Error   : exception renames IO_Exceptions.Mode_Error;
   Name_Error   : exception renames IO_Exceptions.Name_Error;
   Use_Error    : exception renames IO_Exceptions.Use_Error;
   Device_Error : exception renames IO_Exceptions.Device_Error;
   End_Error    : exception renames IO_Exceptions.End_Error;
   Data_Error   : exception renames IO_Exceptions.Data_Error;

private
   ...
end Ada.Direct_IO;
```

Notas:

UNIVERSIDAD DE CANTABRIA

A continuación veremos un ejemplo de entrada/salida directa.

Se usa una función genérica que devuelve una opción elegida en un rango (parámetro genérico) para ser usada en un menú que va a tener tres opciones:

- Mirar_Reserva: devuelve la persona que tiene reservada un aula, leyendo la información del fichero
- Hacer_Reserva: anota en el fichero la reserva de un aula para una persona
- Salir: abandona el programa

El aula se define como un subtipo de los enteros y se usa para acceder directamente a un posición en el fichero.

Ejemplo de Entrada/Salida Directa (1/7)

UNIVERSIDAD DE CANTABRIA

Hacer un programa para anotar en un fichero las reservas de unas aulas

- el aula se identifica por su número
  - será el índice de la casilla del fichero
- la reserva se representa por el nombre de la persona

Daremos las opciones de consultar, hacer reserva y salir, usando el menú genérico:

```
generic
   type Opcion is (<>);
function Menu_Generico return Opcion;
```

Ejemplo de Entrada/Salida Directa (2/7)

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;
function Menu_Generico return Opcion is
   Num_Opcion : Integer;
begin
   -- Poner el menú en pantalla
   New_Line(2);
   Put_Line("--------------Menu------------");
   for Op in Opcion loop
      Put_Line(Integer'Image(Opcion'Pos(Op)+1)&" - "&
               Opcion'Image(Op));
   end loop;
```

Ejemplo de Entrada/Salida Directa (3/7)

UNIVERSIDAD DE CANTABRIA

```
   -- pedir y retornar la opcion deseada
   loop
      begin
         New_Line;
         Put("Introduce opcion deseada : ");
         Get(Num_Opcion); Skip_Line;
         return Opcion'Val(Num_Opcion-1);
      exception
         when Data_Error|Constraint_Error =>
            Skip_Line;
            Put_Line("Error al leer la opcion");
      end;
   end loop;
end Menu_Generico;
```

Ejemplo de Entrada/Salida Directa (4/7)

UNIVERSIDAD DE CANTABRIA

```
with Ada.Direct_Io,Ada.Text_Io, Ada.Integer_Text_IO,
  Var_Strings, Menu_Generico;
use Ada.Text_Io, Ada.Integer_Text_IO, Var_Strings;

procedure Reserva is

   Max_Aulas : constant Integer:=25;

   subtype Aula is Integer range 1..Max_Aulas;
   subtype Persona is Var_String;

   package Reserva_IO is new
     Ada.Direct_IO(Persona);

   procedure Modifica
     (A : Aula; P : Persona; F : in out Reserva_IO.File_Type)
   is
   begin
      Reserva_IO.Write(F,P,Reserva_IO.Count(A));
   end Modifica;
```

Ejemplo de Entrada/Salida Directa (5/7)

UNIVERSIDAD DE CANTABRIA

```
procedure Lee
   (A : Aula; P : out Persona; F : in out Reserva_IO.File_Type) is
begin
    Reserva_IO.Read(F,P,Reserva_IO.Count(A));
end Lee;

procedure Lee_Aula(A : out Aula) is
begin
    loop
       begin
          Put("Aula: ");
          Get(A); Skip_Line;
          exit;
       exception
          when Data_Error|Constraint_Error =>
             Skip_Line;
             Put_Line("Error al leer el aula");
       end;
    end loop;
end Lee_Aula;
```

Ejemplo de Entrada/Salida Directa (6/7)

UNIVERSIDAD DE CANTABRIA

```
   type Opcion is (Mirar_Reserva, Hacer_Reserva, Salir);

   function Pide_Opcion is new Menu_Generico(Opcion);

   -- variables del programa

   F : Reserva_IO.File_Type;
   A : Aula;
   P : Persona;

begin
   begin
      Reserva_IO.Open (F,Reserva_IO.Inout_File,"reserva.dat");
   exception
      when Reserva_IO.Name_Error =>
         Reserva_IO.Create (F,Reserva_IO.Inout_File,"reserva.dat");
         for A in Aula loop
            Modifica(A,To_Var_String("Nadie"),F);
         end loop;
   end;
```

Ejemplo de Entrada/Salida Directa (7/7)

UNIVERSIDAD DE CANTABRIA

```
   loop
      case Pide_Opcion is
         when Mirar_Reserva =>
            Lee_Aula(A);
            Lee(A,P,F);
            Put_Line("La persona es : "&P);
         when Hacer_Reserva =>
            Lee_Aula(A);
            Put("Introduce nombre persona : ");
            Get_Line(P);
            Modifica(A,P,F);
         when Salir =>
           exit;
      end case;
   end loop;
   Reserva_IO.Close(F);
end Reserva;
```

7.7. Resumen de entrada/salida de Streams (1/3)

UNIVERSIDAD DE CANTABRIA

```
with Ada.IO_Exceptions;

package Ada.Streams.Stream_IO is

   type Stream_Access is access all Root_Stream_Type'Class;
   type File_Type is limited private;
   type File_Mode is (In_File, Out_File, Append_File);

   procedure Create
     (File : in out File_Type;
      Mode : in File_Mode := Out_File;
      Name : in String := "";
      Form : in String := "");

   procedure Open
     (File : in out File_Type;
      Mode : in File_Mode;
      Name : in String;
      Form : in String := "");
```

Resumen de entrada/salida de Streams (2/3)

UNIVERSIDAD DE CANTABRIA

```
   procedure Close  (File : in out File_Type);
   procedure Delete (File : in out File_Type);
   procedure Reset  (File : in out File_Type; Mode : in File_Mode);
   procedure Reset  (File : in out File_Type);

   function Mode (File : in File_Type) return File_Mode;
   function Name (File : in File_Type) return String;
   function Form (File : in File_Type) return String;

   function Is_Open     (File : in File_Type) return Boolean;
   function End_Of_File (File : in File_Type) return Boolean;

   function Stream (File : in File_Type) return Stream_Access;

private
   ...
end Ada.Streams.Stream_IO;
```

Resumen de entrada/salida de Streams (3/3)

UNIVERSIDAD DE CANTABRIA

Para leer datos de un tipo determinado:

```
Tipo'Input(Stream:Stream_Access, Dato:Tipo);
```

Para escribir datos de un tipo determinado:

```
Tipo'Output(Stream:Stream_Access, Dato:Tipo);
```

Para que la lectura sea correcta debe hacerse en el mismo orden que la escritura

- Por ejemplo si hemos escrito datos de los tipos `T1`, `T2`, y `T3`, luego deben leerse datos de esos mismos tipos y en ese orden

Notas:

UNIVERSIDAD DE CANTABRIA

En el siguiente ejemplo usamos los streams para hacer entrada/salida de diferentes tipos de datos en el mismo fichero:

- un registro
- un array
- un entero
- un real

Hacemos dos procedimientos de prueba: uno que escribe los datos en un fichero y otro que los recupera de ese mismo fichero.

Observar que el orden en el que se leen los datos debe ser el mismo que en el que se escriben.

Una vez abierto el fichero de streams es necesario usar la función `Stream_IO.Stream` para obtener el puntero sobre el que se aplicarán los atributos `Input` y `Output`.

Ejemplo de entrada/salida de Streams (1/3)

UNIVERSIDAD DE CANTABRIA

```
package Datos_Prueba_Streams is

   type Registro is record
      A,B,C : Integer;
      X : Float;
   end record;

   type Vector_3d is array(1..3) of Float;

end Datos_Prueba_Streams;
```

Ejemplo de entrada/salida de Streams (2/3)

UNIVERSIDAD DE CANTABRIA

```
with Ada.Streams, Ada.Streams.Stream_IO; use Ada.Streams;
with Datos_Prueba_Streams; use Datos_Prueba_Streams;

procedure Prueba_Escribir_Stream is
   F : Stream_IO.File_Type;
   I : Integer:=12;
   X : Float:=32.0;
   R : Registro:=(1,2,3,3.0);
   V : Vector_3d:=(45.0,34.0,-34.0);
   Strm : Stream_IO.Stream_Access;

begin
   Stream_IO.Create(F,Stream_IO.Out_File,"stream.dat");
   Strm:=Stream_IO.Stream(F);
   Integer'Output  (Strm,I);
   Float'Output    (Strm,X);
   Registro'Output (Strm,R);
   Vector_3d'Output(Strm,V);
   Stream_IO.Close(F);
end Prueba_Escribir_Stream;
```

Ejemplo de entrada/salida de Streams (3/3)

UNIVERSIDAD DE CANTABRIA

```
with Ada.Streams, Ada.Streams.Stream_IO; use Ada.Streams;
with Datos_Prueba_Streams; use Datos_Prueba_Streams;

procedure Prueba_Leer_Stream is
   F : Stream_IO.File_Type;
   I : Integer;
   X : Float;
   R : Registro;
   V : Vector_3d;
   Strm : Stream_IO.Stream_Access;

begin
   Stream_IO.Open(F,Stream_IO.In_File,"stream.dat");
   Strm:=Stream_IO.Stream(F);
   I:=Integer'Input  (Strm);
   X:=Float'Input    (Strm);
   R:=Registro'Input (Strm);
   V:=Vector_3d'Input(Strm);
   Stream_IO.Close(F);
end Prueba_Leer_Stream;
```