Curso de
INTRODUCIÓN A LA PROGRAMACIÓN

Resumen del lenguaje de programación Ada (secuencial)

Prof. Javier Campos

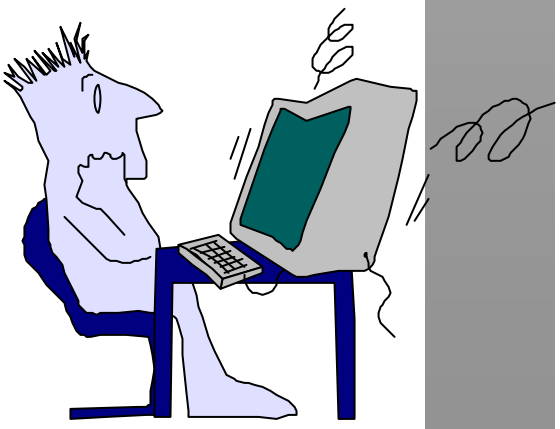
Enero, 2002

Características generales (1…)

- **Lenguaje de propósito general**
- **Profesional (complejo: no pensado para aprendices)**
- **Incorporación de puntos clave de la tecnología de programación:**
  - **Legibilidad**
    - **evitar notación demasiado concisa (es más costoso el mantenimiento que la producción de software: “un programa se lee más veces de las que se escribe”)**
  - **Fuertemente y estáticamente tipado**
    - **gran capacidad para definir datos de tipos diferentes**
    - **cada dato puede usarse sólo en operaciones específicas de su tipo**
    - **la utilización inadecuada se detecta en tiempo de compilación**

Características generales (… y 2)

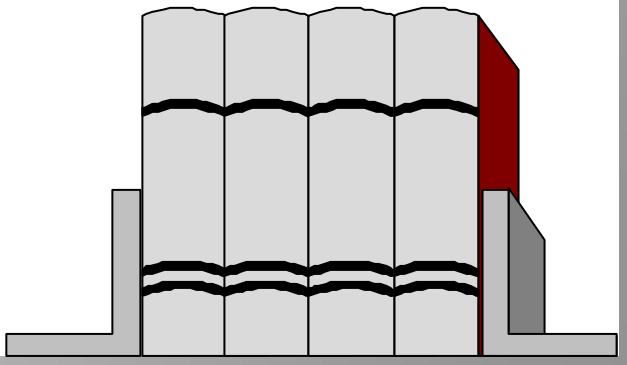
- **Diseño a gran escala**
  - **programación modular**
  - **mecanismos de encapsulación**
  - **compilación separada**
- **Abstracción de datos**
  - **separación clara entre especificación y representación e implementación**
- **Módulos genéricos**
  - **creación de módulos con objetos genéricos (p.e., tipos como parámetros)**
  - **creación de ejemplares de los módulos genéricos para objetos concretos**
- **Programación concurrente**
  - **descripción de procesos que pueden ejecutarse concurrentemente**
  - **definición de operaciones de sincronización y comunicación entre esos procesos**
- **Manejo de excepciones**
  - **definición de comporta-mientos de recuperación ante situaciones de error no previstas**



Bibliografía fundamental

- ***Programming in ADA 95.***
  J.P.G. Barnes.
  Addison-Wesley, 1996
  - **Descripción global del lenguaje**
  - **Estilo didáctico (ejemplos y ejercicios)**
  - **El autor es uno de los miembros clave del equipo de diseño de Ada**

- ***Ada Reference Manual.***
  International Organization for Standarization, ISO/IEC 8652:1995 (e).
  - **Definición del estándar Ada 95**
  - **Imprescindible en el desarrollo de programas (los errores detectados por el compilador hacen referencia concreta a párrafos del libro, como, por ejemplo, [LRM 4.1/3])**



Visión de conjunto: El 1er programa completo

```
-- utilización de módulos estándar de
-- entrada/salida de información textual
with Ada.text_io;
with Ada.integer_text_io;

procedure cuenta_a is
-- lee una secuencia de caracteres terminada
-- con un punto y escribe el nº de veces que
-- ha aparecido la letra 'a‘
  -- para facilitar el uso de módulos
  use Ada.text_io;
  use Ada.integer_text_io;

  -- declaración de constante y variables
  final:constant character := '.';
  un_caracter:character;
  contador:integer:=0;

begin
  put("Introd. sec. terminada por '");
  put(final); put("'"); new_line;
  get(un_caracter);
  while un_caracter/=final loop
    if un_caracter='a'
      then contador:=contador+1;
    end if;
    get(un_caracter);
  end loop;
  put("El número de a's introducidas es");
  put(contador, width => 3); new_line;
end cuenta_a;
```

Palabras reservadas y delimitadores

■ Las 69 palabras reservadas de Ada:

```
abort     declare   goto      out         select
abs       delay                           separate
abstract  delta     if        package     subtype
accept    digits    in        pragma
access    do        is        private     tagged
aliased                       procedure   task
all       else      limited   protected
   terminate
and       elsif     loop                  then
array     end                 raise       type
at        entry     mod       range
          exception           record      until
begin     exit      new       rem         use
body                not       renames
          for       null      requeue     when
case      function            return      while
constant            of        reverse     with
          generic   or
                    others                xor
```

■ Delimitadores simples:

```
& ' ( ) * + , - . / : ; < = > |
```

■ Delimitadores compuestos:

`=>`	se usa en registros, instrucciones 'case', etc.
`..`	para rangos
`**`	potenciación
`:=`	asignación
`/=`	desigualdad
`>=`	mayor o igual que
`<=`	menor o igual que
`<<`	paréntesis de etiquetas (para 'gotos')
`>>`	el otro paréntesis de etiquetas
`<>`	se usa en vectores y módulos genéricos

Tipos escalares: Generalidades

Declaraciones de objetos

```
pi:constant float:=3.1416;
final:constant character:='.';
i,j,k:integer;
p,q:integer:=12;
```

Ambito y visibilidad - Pascal

Tipos

- tipo = conjunto de valores y de operadores
- no se puede asignar valor a una variable de un tipo diferente (Ada es fuertemente tipado)

```
type integer is … ;   -- predefinido
```

Subtipos

- Sirve para caracterizar un subconjunto de los valores de un tipo
- NO constituye un nuevo tipo (la asignación está permitida)

```
subtype dia is integer range 1..31;
subtype dia_feb is dia range 1..29;
d1:dia;
d2:dia_feb;
```

```
d3:integer range 1..31;
```

Tipos escalares: Discretos (1…)

■ Tipos definidos por enumeración

- **Hay dos predefinidos (booleanos y caracteres)**

```
type dia is (lunes,martes,miercoles,
      jueves,viernes,sabado,domingo);
subtype laborable is dia range
                        lunes..viernes;

d1:dia; d2:laborable;
```

- **Atributos:**

```
dia'first=lunes
dia'last=domingo
dia'succ(lunes)=martes
dia'pred(martes)=lunes
dia'pos(lunes)=0
dia'val(1)=martes
dia'image(lunes)="LUNES"
dia'value("martes")=martes
```

- **Operadores relacionales:**

```
=, /=, <, <=, >, >=
```

- **Comprobadores de pertenencia :**

```
d1 in laborable
d2 not in lunes..miercoles
```

Tipos escalares: Discretos (2)

■ Tipo booleano (predefinido):

```
type boolean is (false,true);
```

```
a,b,c,d:boolean;

((not a) and b) or (c xor d)
```

■ Tipo carácter

➢ **Conjunto de valores (código ASCII)**

'€'	'€'	'€'	'€'	'€'	'€'	'€'	'€'
'€'	'€'	'€'	'€'	'€'	'€'	'€'	'€'
'€'	'€'	'€'	'€'	'€'	'€'	'€'	'€'
'€'	'€'	'€'	'€'	'€'	'€'	'€'	'€'
' '	'!'	'"'	'#'	'$'	'%'	'&'	'''
'('	')'	'*'	'+'	','	'-'	'.'	'/'
'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'
'8'	'9'	':'	';'	'<'	'='	'>'	'?'
'@'	'A'	'B'	'C'	'D'	'E'	'F'	'G'
'H'	'I'	'J'	'K'	'L'	'M'	'N'	'O'
'P'	'Q'	'R'	'S'	'T'	'U'	'V'	'W'
'X'	'Y'	'Z'	'['	'\'	']'	'^'	'_'
'`'	'a'	'b'	'c'	'd'	'e'	'f'	'g'
'h'	'i'	'j'	'k'	'l'	'm'	'n'	'o'
'p'	'q'	'r'	's'	't'	'u'	'v'	'w'
'x'	'y'	'z'	'{'	'|'	'}'	'~'	'•'

'€' **representan caracteres de control**

➢ **Operadores: los de cualquier tipo definido por enumeración**

Tipos escalares: Discretos (… y 3)

■ Tipos enteros

- ➢ **Hay algunos predefinidos:**

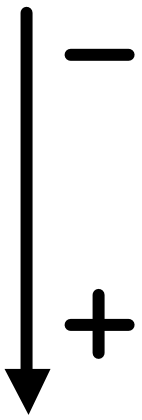
```
type integer is … ;
type short_integer is … ;
type long_integer is … ;
subtype natural is integer range
                        0..integer'last;
subtype positive is integer range
                        1..integer'last;
```

- ➢ **Pueden definirse otros (ver bibliografía)**
- ➢ **Operadores específicos (además de los introducidos para los tipos enumerados):**
  - – **monoarios**

    `+, -, abs`
  - – **binarios**

    `+, -, *, /, rem, mod, **`
- ➢ **Prioridades:**

```
and, or, xor
not
=, /=, <, <=, >, >=, in, not in
+, -    (binarios)
+, -    (monoarios)
*, /, mod, rem
**, abs
```



Tipos escalares: Reales

■ Tipos reales

- ➢ **Hay varios (coma flotante y coma fija)**
- ➢ **Tipo predefinido estándar (coma flotante):**

```
type float is … ;
```

- ➢ **Definición exigiendo una precisión:**

```
type mis_reales is digits 7;
```

- ➢ **Operadores específicos**
  - – **monoarios**

    `+, -, abs`
  - – **binarios**

    `+, -, *, /, **`

- ➢ **No se admite aritmética mixta:**

```
n:integer; x:float;

n+x

float(n)+x
n+integer(x)
```

¡ INCORRECTO !

Correcto

La conversión de real a entero efectúa redondeo.

Estructuración del control: Condicionales

■ Instrucciones condicionales

```
if … then … end if;
```

```
if …
  then …
  else …
end if;
```

```
if    … then …
elsif … then …
elsif … then …
else          …
end if;
```

```
case … is
  when     …    => …
  when  …|…|…  => …
  when   …..…  => …
  when     …   => null;
  when others => …
end case;
```

Estructuración del control: Bucles y bloques

■ Instrucciones iterativas

```
while … loop
  …
end loop;
```

```
for d in dia loop … end loop;
```

```
for d in lunes..viernes loop
  …
end loop;
```

```
for d in reverse lunes..viernes loop
  …
end loop;
```

```
loop
  …
end loop;
```

```
loop
  …; exit; …;
end loop;
```

```
loop
  …
exit when …;
  …
end loop;
```

■ Bloques

➢ **bloque = instrucción con objetos locales**

```
declare
  aux:integer:=a;
begin
  a:=b; b:=aux;
end;
```

Estructuración del control: Subalgoritmos (1...)

■ Procedimientos y funciones

➢ Paso de parámetros:

```
in, out, in out
```

```
procedure toto(x:T1; y:in out T2) is
  …
end toto;
```

```
toto(e,z);          -- llamada normal
toto(y=>z,x=>e);    -- llamada nombrada
```

```
procedure titi(x:T1;
               y:in integer:=3;
               d:in dia:=lunes) is
  …
end titi;
```

```
titi(e);  -- valores por defecto
titi(e,24,martes);
titi(x=>e,d=>jueves);
```

➢ Funciones

```
function factorial(n:natural)
                   return natural is
begin
  if n in 0..1
    then return 1;
    else return n*factorial(n-1);
  end if;
end factorial;
```

```
put(factorial(8));
```

Estructuración del control: Subalgoritmos (…y 2)

- **Sobrecarga de operadores**

  Los siguientes operadores

```
and  or   xor
=    <    <=   >    >=
+    -    &    abs  not
*    /    mod  rem  **
```

  pueden redefinirse.

```
function "*"(u,v:vector)
             return float is
  resultado:float:=0.0;
begin
  for i in u'range loop
    resultado:=resultado+u(i)*v(i);
  end loop;
  return resultado;
end "*";
```

```
u,v:vector;
...
x:=u*v;   -- producto escalar
```

  El significado se distingue por el contexto.

  No se pueden modificar la aridad ni la sintaxis de llamada (prefija o infija).

Tipos compuestos: Vectores (1...)

■ Vectores

➢ Definiciones restringidas

```
type t1 is array(1..10) of boolean;
type t2 is array(dia) of t1;
type t3 is
  array(lunes..jueves,-10..14) of t1;

x:t1; y:t2; z:t3;
```

```
x(i+j)
y(martes)
y(martes)(i+j)
z(martes,i+j)
```

```
x(5..8)
y(martes..viernes)
```

rebanadas
(vectores de 4 comp.)

➢ Definiciones no restringidas

caja

```
type matriz is
    array(positive range <>,
          positive range <>) of real;

function "+"(a,b:matriz)
             return matriz is
begin … end "+";
```

```
subtype mat23 is matriz(1..2,1..3);
m1,m2,m3:mat23;

m3:=m1+m2;
```

Tipos compuestos: Vectores (2)

- **Atributos relacionados con los índices**

```
function "+"(a,b:matriz)
              return matriz is
  suma:matriz(a'range(1),a'range(2));
begin
  for i in a'range(1) loop
    for j in a'range(2) loop
      suma(i,j):=a(i,j)+b(i,j);
    end loop;
  end loop;
  return suma;
end "+";
```

Otros atributos: `'first, 'last, 'length`

- **Constantes de tipos vectoriales**

```
m1:mat23:=((1.0,2.0,3.0),
           (4.0,5.0,6.0));
```

```
siguiente:constant array(dia) of dia
    :=(martes,miercoles,jueves,
       viernes,sabado,domingo,lunes);
```

```
m1:=((1..3=>1.0),
     (1=>2.0,2=>3.0,3=>4.0));
```

```
m2:=mat23'(1=>(1=>1.0,others=>0.0),
           2=>(2=>1.0,others=>0.0));
```

¡ Atención: 'others' exige límites conocidos !

Tipos compuestos: Vectores (… y 3)

➢ **Tipo no restringido predefinido: string**

```
type string is array(positive range <>)
                        of character;
```

```
x0:string(1..8);
subtype s1 is string(2..8);
subtype linea is string(1..80);
x1:s1;
l:linea;
```

```
l(3..10):=x0;
l(2..4):=x0(4..6);
l(1..8):=x0(1..4) & "A" & x1(2..4);
```

las constantes de tipos cadena se escriben entre comillas

¡ Ojo ! ¡ `x0` es una cadena de caracteres de longitud exactamente `8` !

(NO es una cadena de longitud menor o igual que `8`)

Tipos compuestos: Registros

■ Registros

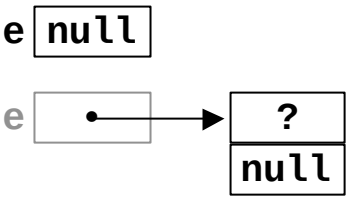
```
type nombre_mes is
  (ene,feb,mar,abr,may,jun,jul,ago,
   sep,oct,nov,dic);

type fecha is
  record
    dia:integer range 1..31;
    mes:nombre_mes;
    anyo:integer:=1994;
  end record;

descubrimiento:constant fecha(12,oct,
                              1492);
ayer,hoy:fecha;

ayer:=(mes=>ago,dia=>8,anyo=>1994);
hoy:=ayer; hoy.dia:=hoy.dia+1;
```

- **Un registro NO puede tener campos de tipo vector no restringido**

- **Hay otras formas de definir tipos registro, por ejemplo, incluyendo partes variantes (ver bibliografía)**

Tipos puntero y datos dinámicos (1…)

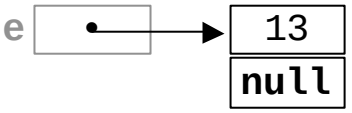
```
type celda;
type enlace is access celda;
type celda is
  record
    valor:integer;
    siguiente:enlace;
  end record;

e:enlace;
```

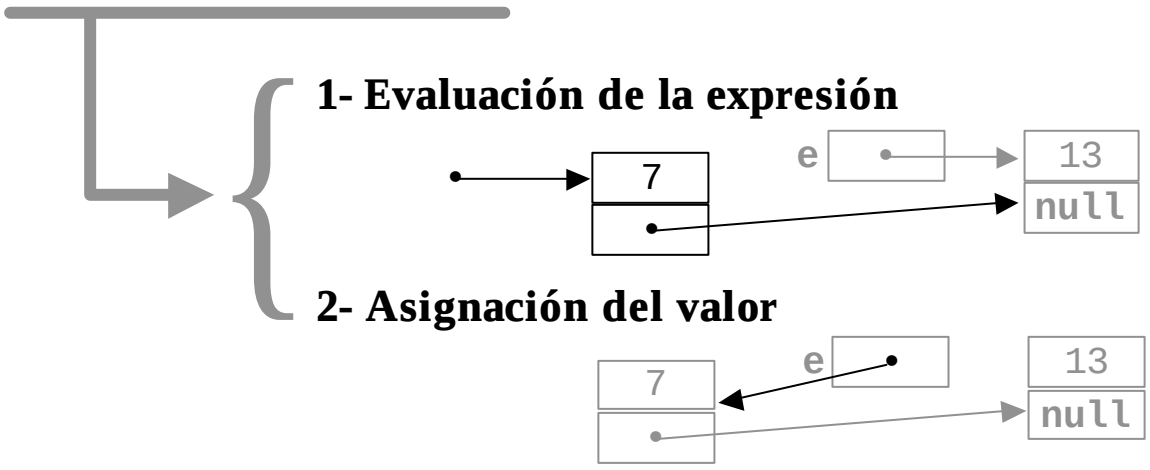



```
e:=new celda;


e.valor:=13;
```



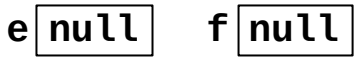
```
e:=new celda'(13,null);


e:=new celda'(7,e);
```



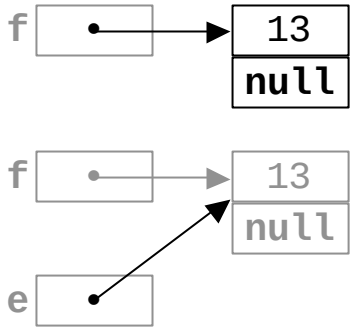


Tipos puntero y datos dinámicos (… y 2)

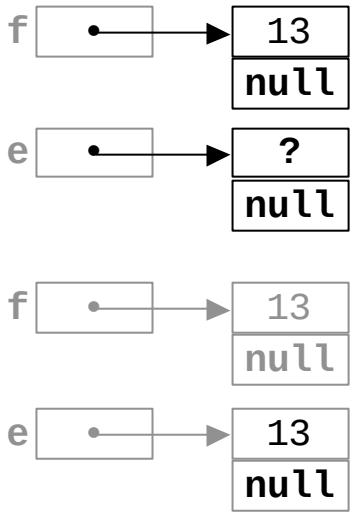
```
e,f:enlace;
```




```
f:=new celda'(13,null);


e:=f;
```




```
f:=new celda'(13,null);
e:=new celda;




e.all:=f.all;
```



- **Liberación de memoria de datos inaccesibles: suele proporcionarla la implementación.**
- **También puede hacerlo el programador:**

```
with unchecked_deallocation;

procedure disponer is new
    unchecked_deallocation(celda,enlace);

e:enlace;
...
disponer(e);
```

Estructura global: Módulos (1…)

■ Módulos

➢ Módulo de declaración:

```
package conjuntos is

  type conjcar is private;

  procedure vacio (A:out conjcar);
  function  esVacio (A:in conjcar)
                     return boolean;
  procedure poner (c:in character;
                   A:in out conjcar);
  procedure quitar (c:in character;
                    A:in out conjcar);
  function  pertenece (c:in character;
                       A:in conjcar)
                      return boolean;
  procedure union (A,B:in conjcar;
                   C:out conjcar);
  procedure interseccion (A,B:in conjcar;
                          C:out conjcar);
  function cardinal (A:in conjcar)
                     return integer;
private

  type elementos is array(character)
                            of boolean;
  type conjcar is
    record
      elmto:elementos;
      card:integer;
    end record;

end conjuntos;
```

limited

Estructura global: Módulos (2)

- **Módulo de implementación:**

```
package body conjuntos is

  procedure vacio (A:out conjcar) is
  begin
    A.card:=0;
    for c in character loop
      A.elmto(c):=false;
    end loop;
  end vacio;

  function  esVacio (A:in conjcar)
                    return boolean is
  begin
    return A.card=0;
  end esVacio;

  function  pertenece (c:in character;
                       A:in conjcar)
                      return boolean is
  begin
    return A.elmto(c);
  end pertenece;

  procedure poner (c:in character;
                   A:in out conjcar) is
  begin
    if not pertenece(c,A) then
      A.elmto(c):=true; A.card:=A.card+1;
    end if;
  end poner;

  ...
end conjuntos;
```

Estructura global: Módulos (… y 3)

- **Utilización de los objetos de un módulo**

```
procedure toto is
  package conjuntos is  -- definición e
    ...                 -- implementación
  end conjuntos;

  A,B,C:conjuntos.conjcar;

begin
  ...
  conjuntos.vacio(A);
  ...
  conjuntos.union(A,B,C);
  ...
end toto;
```

```
procedure toto is
  package conjuntos is  -- definición e
    ...                 -- implementación
  end conjuntos;

  use conjuntos;

  A,B,C:conjcar;

begin
  ...
  vacio(A);
  ...
  union(A,B,C);
  ...
end toto;
```

Estructura global: Compilación separada (1...)

Compilación separada

- **Compilación única:**

```
procedure toto is
  package lala is ... end lala;
  ...
  package body lala is ... end lala;
begin
  ...
end toto;
```

- **Compilación separada subordinada:**

Primera compilación

```
procedure toto is
  package lala is ... end lala;
  ...
  package body lala is separate;
begin
  ...
end toto;
```

Segunda compilación

```
separate(toto)
package body lala is ... end lala;
```

Estructura global: Compilación separada (... y 2)

➢ **Compilación separada no subordinada:**

Primera compilación

```
package lala is ... end lala;
```

Segunda compilación

```
package body lala is ... end lala;
```

Tercera compilación

```
with lala;
procedure toto is
   ...
begin
   ...
end toto;
```

Módulos y programas genéricos (1…)

■ Algoritmos genéricos

Tipos como parámetro

```
generic
   type item is private;
procedure canjear(x,y:in out item);

procedure canjear(x,y:in out item) is
   aux:item;
begin
   aux:=x;
   x:=y;
   y:=aux;
end;
```

```
procedure swap is new canjear(float);
procedure swap is new canjear(character);
```

Módulos y programas genéricos (2)

■ Algoritmos genéricos (cont.)

```
generic
  type ind is (<>);
  type elem is private;
  type vector is array(ind range <>)
                       of elem;
  with function ">"(a,b:elem)
                    return boolean;
package ordenacion_g is
  procedure ordena(v:in out vector);
end;
package body ordenacion_g is
  procedure ordena(v:in out vector) is
    i,j:ind; m,t:elem; n:integer;
  begin
    ...
  end ordena;
end ordenacion_g;
```

```
with ordenacion_g;
procedure titi is
  type color is (rojo,azul,gris);
  type dia is (lu,ma,mi,ju,vi,sa,do);
  type vect is array(day range <>)
                     of color;
  x:vect(ma..vi):=(gris,azul,rojo,gris);
  package o is
    new ordenacion_g(dia,color,vect,">");
begin
  ...
  o.ordena(x);
  ...
end;
```

Módulos y programas genéricos (… y 3)

■ Módulos genéricos: TAD's genéricos

```
generic
  type base is (<>);
package conjuntos is
  type conjunto is private;
  procedure vacio (A:out conjunto);
  function  esVacio (A:in conjunto)
                     return boolean;
  procedure poner (c:in base;
                   A:in out conjunto);
  ...
private
  type elementos is array(base)
                           of boolean;
  type conjunto is
    record
      elmto:elementos;
      card:integer;
    end record;
end conjuntos;
```

➢ **Otras especificaciones de tipos genéricos:**

```
type t is private;
type t is limited private;
type t is (<>);
type t is range <>;
type t is digits <>;
type t is delta <>;
```

- **tipo con ':=' y '='**
- **tipo sin ':=' ni '='**
- **tipo discreto**
- **tipo entero**
- **tipo coma flotante**
- **tipo coma fija**

Entradas/salidas (1…)

■ Ficheros de acceso secuencial

```
with io_exceptions;
generic
  type element_type(<>) is private;
package sequential_io is
  type file_type is limited private;
  type file_mode is
                (in_file,out_file,append_file);

  -- Gestión de ficheros

  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 form(file:in file_type)
                return string;
  function is_open(file:in file_type)
                   return boolean;
  ...
```

Entradas/salidas (2)

```
   ...

   -- Operaciones de entrada/salida

   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;

   -- Excepciones

   ...

private
   -- Dependiente de la implementación
end sequential_io;
```

- **Hay otro módulo similar para ficheros de acceso directo**

Entradas/salidas (3)

■ Ficheros de texto

```
with io_exceptions;
package text_io is
  type file_type is limited private;
  type file_mode is
              (in_file,out_file,append_file);

  -- Nº de columna en una línea y de línea
  -- en una página, y formatos

  type count is range 0..dep_implementación;
  subtype positive_count is
                  count range 1..count'last;
  unbounded:constant count:=0;

  subtype field is
        integer range 0..dep_implementación;
  subtype number_base is integer range 2..16;
  type type_set is (lower_case,upper_case);

  -- Gestión de ficheros

  -- create, open, close, delete, reset,
  -- mode, name, form e is_open son iguales
  -- que en sequential_io

  -- Control de ficheros de e/s por defecto

  procedure set_output(file:in file_type);
  function standard_output return file_type;
  function current_output return file_type;

  -- Lo mismo para entrada, con 'input'
  ...
```

Entradas/salidas (4)

```
...

-- Especificación de longitudes de línea y
-- página

procedure set_line_length(to:in count);
procedure set_page_length(to:in count);
function line_length return count;
function page_length return count;

-- Lo mismo con un parámetro file

-- Control de columnas, líneas y páginas

procedure new_line(spacing:in
                        positive_count:=1);
procedure skip_line(spacing:in
                        positive_count:=1);
function end_of_line return boolean;
procedure new_page;
procedure skip_page;
function end_of_page return boolean;
function end_of_file return boolean;
procedure set_col(to:in positive_count);
procedure set_line(to:in positive_count);
function col return positive_count;
function line return positive_count;
function page return positive_count;

-- Lo mismo con un parámetro file

...
```

Entradas/salidas (5)

```
...

-- Entrada/salida de caracteres

procedure get(item:out character);
procedure put(item:in character);

-- Lo mismo con un parámetro file

-- Entrada/salida de cadenas de caracteres

procedure get(item:out string);
procedure put(item:in string);

procedure get_line(item:out string;
                   last:out natural);
procedure put_line(item:in string);

-- Lo mismo con un parámetro file

...
```

Entradas/salidas (6)

```
...

-- Módulo genérico de e/s de enteros

generic
  type num is range <>;
package integer_io is
  default_width:field:=num'width;
  default_base:number_base:=10;

  procedure get(item:out num;
                width:in field:=0);
  procedure put(item:in num;
                width:in field:=
                             default_width;
                base:in number_base:=
                             default_base);

  -- Lo mismo con un parámetro file

  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;

...
```

Entradas/salidas (7)

```
...

-- Módulo genérico de e/s de tipos reales
-- (de coma flotante)

generic
  type num is digits <>;
package float_io is
  default_fore:field:=2;
  default_aft:field:=num'digits-1;
  default_exp:field:=3;

  procedure get(item:out num;
                width:in field:=0);
  procedure put(item:in num;
                fore:in field:=default_fore);

  -- Lo mismo con un parámetro file

  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;

...
```

Entradas/salidas (… y 8)

```
  ...

  -- Módulo genérico de e/s de tipos
  -- definidos por enumeración

  generic
    type enum is (<>);
  package enumeration_io is
    default_width:field:=0;
    default_setting:type_set:=upper_case;

    procedure get(item:out enum);
    procedure put(item:in enum;
                  width:in field:=
                              default_width;
                  set:in type_set:=
                            default_setting);

    -- Lo mismo con un parámetro file

    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;

  -- Excepciones
  ...

private
  -- Dependiente de la implementación
end text_io;
```