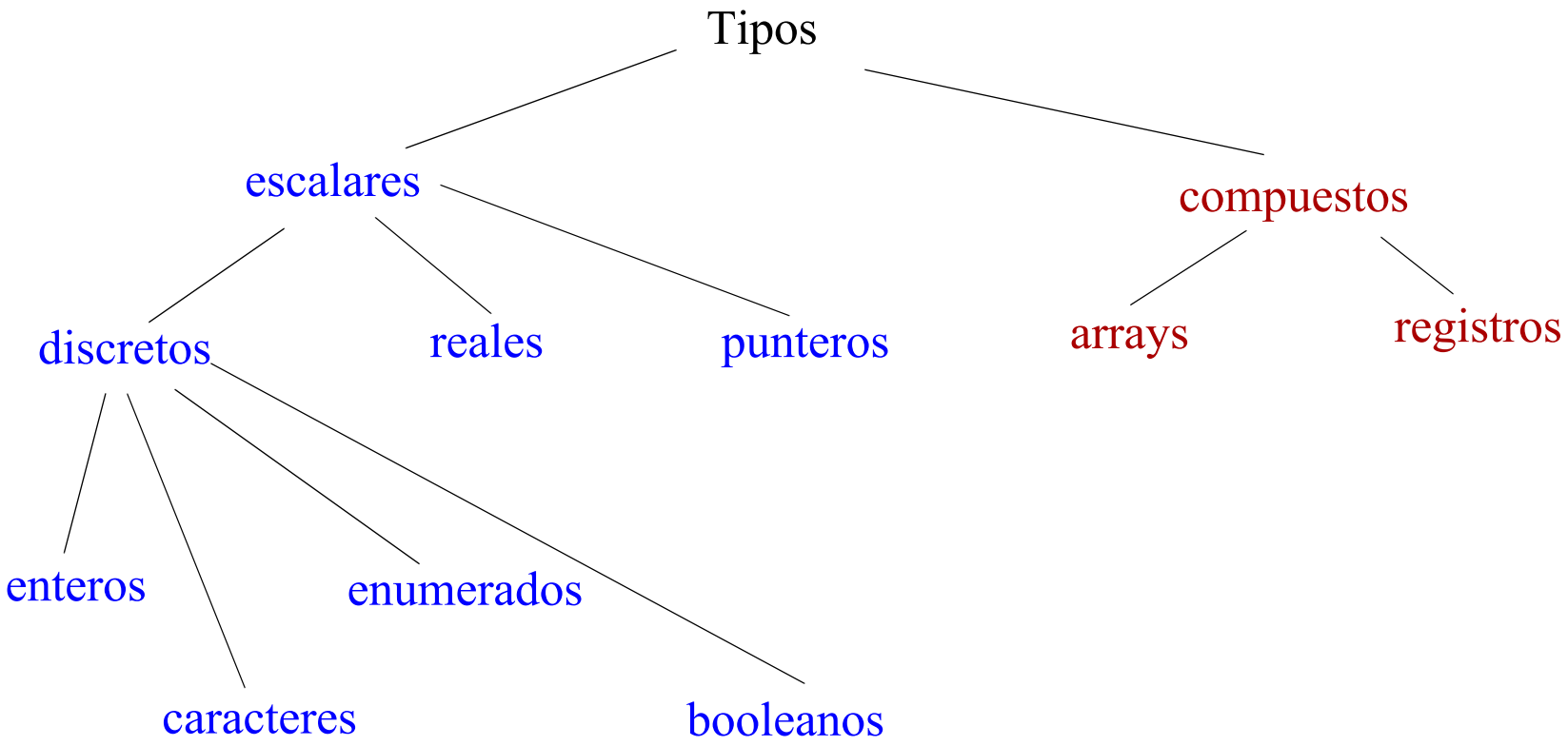
Tipos de datos

UNIVERSIDAD DE CANTABRIA

Jerarquía de tipos:



Declaración de tipos

UNIVERSIDAD DE CANTABRIA

Formato:

```
type Nombre_Tipo is definicion;
```

Ojo: un tipo de datos no es un objeto de datos:

- no ocupa espacio en memoria
- es sólo una definición para usar más adelante al crear variables y constantes

Concepto de *subtipo*: un tipo de datos puede tener subtipos, que restringen el rango o la precisión del tipo.

- Los datos de diferentes tipos no se pueden mezclar.
- Pero podemos mezclar datos de diferente subtipo si proceden del mismo tipo.

Tipos enteros

UNIVERSIDAD DE CANTABRIA

Declaración:

type Nombre is range valor_inicial .. valor_final;

Subtipos enteros:

subtype Nombre is Tipo range valor_inicial .. valor_final;

Ejemplo: declaraciones de datos en un prog. de nóminas

UNIVERSIDAD DE CANTABRIA

```
type Dinero is range 0..1_000_000; -- euros

type Sueldo_Director  is range 0..100_000;
type Sueldo_Ingeniero is range 0..10_000;
type Sueldo_Becario   is range 0..500; -- pobrecillo

D : Dinero;
S_D : Sueldo_Director;
S_I : Sueldo_Ingeniero;
S_B : Sueldo_Becario;
```

Para calcular la nómina si hay 1 director, 3 ingenieros y cuatro becarios:

```
D:=S_D+3*S_I+4*S_B; -- mal, pues no se pueden meclar tipos
D:=Dinero(S_D)+3*Dinero(S_I)+4*Dinero(S_B); -- bien, pero largo
```

Uso de subtipos

UNIVERSIDAD DE CANTABRIA

Puesto que los sueldos y el dinero son de la misma naturaleza, es mejor usar subtipos:

```
type Dinero is range 0..1_000_000; -- euros

subtype Sueldo_Director  is Dinero range 0..100_000;
subtype Sueldo_Ingeniero is Dinero range 0..10_000;
subtype Sueldo_Becario   is Dinero range 0..500;

D : Dinero;
S_D : Sueldo_Director;
S_I : Sueldo_Ingeniero;
S_B : Sueldo_Becario;
```

Y ahora para calcular la nómina:

```
D:=S_D+3*S_I+4*S_B; -- correcto, pues todos son del mismo tipo
```

Más sobre subtipos

UNIVERSIDAD DE CANTABRIA

¿Cuándo usar tipos o subtipos?

- usar subtipos para cosas de la misma naturaleza (p.e., sueldos)
- usar tipos para cosas de naturaleza diferente (p.e., sueldo y temperatura)

Subtipos enteros predefinidos:

```
subtype Natural  is Integer range 0..Integer'Last;
subtype Positive is Integer range 1..Integer'Last;
```

Tipos Reales

UNIVERSIDAD DE CANTABRIA

Declaración:

```
type Tipo is digits n;
type Tipo is digits n range val1..val2;
```

Subtipos reales:

```
subtype Nombre is Tipo range val1..val2;
```

Tipos enumerados

UNIVERSIDAD DE CANTABRIA

Sus valores son identificadores. Evitan la necesidad de usar "códigos".

```
type Color is (Rojo, Verde, Azul);
type Escuela is (Teleco, Caminos, Fisicas);
```

Los valores están ordenados, por el orden en que se escriben

Atributos más útiles de los tipos enumerados:

Tipo'First primer valor
Tipo'Last último valor
Tipo'Succ(Valor) sucesor: siguiente a Valor
Tipo'Pred(Valor) predecesor
Tipo'Pos(Valor) código numérico del Valor (empiezan en cero)
Tipo'Val(Número) Valor enumerado correspondiente al número
Tipo'Image(Valor) Conversión a texto
Tipo'Value(Texto) Conversión de texto a enumerado

Entrada/salida de tipos escalares

UNIVERSIDAD DE CANTABRIA

`Ada.Text_IO` contiene submódulos genéricos:

- los podemos especializar para leer o escribir datos de tipos creados por nosotros

Poner en las declaraciones una (o varias) de estas líneas:

```
package Int_IO   is new Ada.Text_IO.Integer_IO(Mi_Tipo_Entero);
package F_IO     is new Ada.Text_IO.Float_IO(Mi_Tipo_Real);
package Color_IO is new Ada.Text_IO.Enumeration_IO (Mi_Tipo_Enum);
```

Esto crea los módulos `Int_IO`, `F_IO`, y `Color_IO`

- cada uno con operaciones `Get` y `Put` para leer o escribir, respectivamente, datos de los tipos indicados.
- tratamiento igual que con el resto de módulos (ej. cláusula `use`)

Ejemplo con tipos subrango y enumerados

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_IO;
use Ada;
procedure Nota_Media_Enum is
     type Nota_num is range 0..10;
     type Nota_Letra is
       (Suspenso,Aprobado,Notable,Sobresaliente);
     package Nota_IO is new
             Text_IO.Integer_IO(Nota_Num);
     package Letra_IO is new
             Text_IO.Enumeration_IO(Nota_Letra);
     Nota1,Nota2,Nota3,Nota_Media : Nota_Num;
     Nota_Final : Nota_Letra;
begin
     Text_IO.Put("Nota del primer trimestre: ");
     Nota_IO.Get(Nota1);
     Text_IO.Skip_Line;
     Text_IO.Put("Nota del segundo trimestre: ");
     Nota_IO.Get(Nota2);
```

Ejemplo con tipos subrango y enumerados (cont.)

UNIVERSIDAD DE CANTABRIA

```
    Text_IO.Skip_Line;
    Text_IO.Put("Nota del tercer trimestre: ");
    Nota_IO.Get(Nota3);
    Text_IO.Skip_Line;
    Nota_Media := (Nota1+Nota2+Nota3)/3;
    Text_IO.Put("Nota Media : ");
    Nota_IO.Put(Nota_Media);
    Text_IO.New_Line;
    case Nota_Media is
        when 0..4  => Nota_Final:=Suspenso;
        when 5..6  => Nota_Final:=Aprobado;
        when 7..8  => Nota_Final:=Notable;
        when 9..10 => Nota_Final:=Sobresaliente;
    end case;
    Text_IO.Put("Nota Final : ");
    Letra_IO.Put(Nota_Final);
    Text_IO.New_Line;
end Nota_Media_Enum;
```

A destacar

UNIVERSIDAD DE CANTABRIA

Las cláusulas `use` permitirían reducir el texto

Los tipos subrango permiten detectar errores de rango de manera automática

- más adelante aprenderemos a tratar estos errores

El tipo subrango permite omitir la cláusula `others` en la instrucción case

Arrays

UNIVERSIDAD DE CANTABRIA

Permiten almacenar muchos datos del mismo tipo: tablas o listas de valores, vectores, etc.

Pueden ser multidimensionales: matrices,...

Su tamaño no puede cambiar después de crearlo

Se identifican por un nombre y un rango de valores (índice) que debe ser discreto

Declaración:

```
type Nombre is array (rango) of Tipo_Elemento;
```

Ejemplos

UNIVERSIDAD DE CANTABRIA

```
type Vector_3D is array (1..3) of Float;

type Meses is (Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio,
               Agosto, Septiembre, Octubre, Noviembre, Diciembre);
type Num_Dias_Mes is array (Integer range 1..12) of Natural;
type Num_Dias is array (Meses) of Natural;

type Matriz is array (1..3,1..3) of Float;

subtype Hora is Integer range 0..23;
subtype Temp is Float range -273.0..1000.0;
type Tabla_Temperaturas is array (Hora,Meses) of Temp;
```

Los rangos no tienen por qué ser estáticos:

```
N : Integer := valor;
type Bool is array (1..N) of Boolean;
```

Uso del array

- Completo: por su nombre
- Un elemento: `nombre(indice)`
- Una parte: `nombre(indice1..indice2)`

Ejemplos:

```
V1, V2   : Vector_3D
M        : Matriz
Dias     : Num_Dias;
T        : Tabla_Temperaturas;
B        : array (1..100) of Integer; -- Array de tipo anónimo
Contactos : Bool;
...
V1(1):=3.0+V2(3);
V2:=V1;
M(2,3):=M(1,1)*2.0;
Dias(Enero):=31;
T(20,Febrero):=23.1;
```

Atributos de arrays

UNIVERSIDAD DE CANTABRIA

`Tipo'Range` o `Array'Range`: es el rango de valores

```
for i in V1'Range loop
   V1(i):=0.0;
end loop;
```

Literales de array:

```
V1:=(0.0, 3.2, 1.2);
Contactos:=Bool'(1..3 => True, 4 => False,
              5|8 => True, others =>False);
M:= Matriz'(1..3 => (1..3 => 0.0));
Dias:=(31,28,31,30,31,30,31,31,30,31,30,31);
Dias:=Num_Dias'(febrero =>28,
              abril|junio|septiembre|noviembre =>30,
              others =>31);
B(1..4):=(0,1,2,3);
```

Ejemplo de programa que usa vectores

UNIVERSIDAD DE CANTABRIA

Cálculo del producto escalar de dos vectores de dimensión definible por el usuario:

- **Modalidad 1: usar parte de un array grande**
  - una variable entera almacena el tamaño útil
- **Modalidad 2: crear el array del tamaño justo**
- **Modalidad 3: usar arrays no restringidos**

Ejemplo versión 1: arrays de dimensión fija

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_IO,Ada.Integer_Text_IO,Ada.Float_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO, Ada.Float_Text_IO;

procedure Producto is
   Dimension_Max : constant Integer := 100;
   type Vector is array (1..Dimension_Max) of Float;
   V1,V2         : Vector;
   N             : Integer range 1..Dimension_Max;
   Prod_Escalar  : Float:=0.0;
begin
   Put("Introduce dimension : ");
   Get(N);
   Skip_Line;
   Put_Line("Vector V1:");
   for I in 1..N loop
      Put("Introduce componente ");
      Put(I); Put(": ");
      Get(V1(I)); Skip_Line;
   end loop;
```

Ejemplo versión 1: arrays de dimensión fija (cont.)

UNIVERSIDAD DE CANTABRIA

```
    Put_Line("Vector V2:");
    for I in 1..N loop
       Put("Introduce componente ");
       Put(I);
       Put(": ");
       Get(V2(I));
       Skip_Line;
    end loop;

    for I in 1..N loop
       Prod_Escalar:=Prod_Escalar + V1(I)*V2(I);
    end loop;
    Put("El producto escalar es : ");
    Put(Prod_Escalar);
    New_Line;
end Producto;
```

Arrays de tamaño variable

UNIVERSIDAD DE CANTABRIA

Modalidad 2: declarar los arrays del tamaño justo

Usar para ello la instrucción `declare`

- Introduce un nuevo bloque donde realizar declaraciones dentro de un procedimiento o función

```
leer N;
declare
  v1,v2 : array(1..N) of Float;
begin
  ...-- uso de v1 y v2
end;
```

Ejemplo versión 2: arrays de dimensión variable

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_Io,Ada.Integer_Text_Io,Ada.Float_Text_Io;
use Ada.Text_Io, Ada.Integer_Text_Io, Ada.Float_Text_Io;

procedure Producto_Variable is
   N             : Positive;
   Prod_Escalar  : Float:=0.0;
begin
   Put("Introduce dimension : ");
   Get(N);
   Skip_Line;
   declare
      type Vector is array (1..N) of Float;
      V1,V2 : Vector;
   begin
      Put_Line("Vector V1:");
      for I in Vector'Range loop
         Put("Introduce componente ");
         Put(I);
         Put(": ");
```

Ejemplo versión 2: arrays de dimensión variable (cont.)

UNIVERSIDAD DE CANTABRIA

```
            Get(V1(I)); Skip_Line;
         end loop;
         Put_Line("Vector V2:");
         for I in V2'Range loop
            Put("Introduce componente ");
            Put(I);
            Put(": ");
            Get(V2(I)); Skip_Line;
         end loop;

         for I in 1..N loop
            Prod_Escalar:=Prod_Escalar + V1(I)*V2(I);
         end loop;
         Put("El producto escalar es : ");
         Put(Prod_Escalar);
         New_Line;
      end;
end Producto_Variable;
```

Arrays no restringidos

UNIVERSIDAD DE CANTABRIA

```
type Vector is array (Integer range <>) of Float;

V1 : Vector(1..100);
V2 : Vector(1..200);
```

Modalidad 3: usar un tipo irrestringido y declarar los arrays del tamaño justo

```
type Vector is array(Integer range <>) of Float;

leer N
declare
  v1,v2 : Vector(1..N);
begin
  ...-- uso de v1 y v2
end;
```

Ejemplo versión 3: arrays no restringidos

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_IO,Ada.Integer_Text_IO,Ada.Float_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO, Ada.Float_Text_IO;

procedure Producto_Irrestringido is
   type Vector is array (Positive range <>) of Float;
   N             : Positive;
   Prod_Escalar : Float:=0.0;
begin
   Put("Introduce dimension : ");
   Get(N);
   Skip_Line;
   declare
      V1,V2          : Vector(1..N);
   begin
      Put_Line("Vector V1:");
      for I in V1'Range loop
         Put("Introduce componente ");
         Put(I);
         Put(": ");
```

Ejemplo versión 3: arrays no restringidos (cont.)

UNIVERSIDAD DE CANTABRIA

```
            Get(V1(I));
            Skip_Line;
        end loop;
        Put_Line("Vector V2:");
        for I in V2'Range loop
            Put("Introduce componente ");
            Put(I);
            Put(": ");
            Get(V2(I)); Skip_Line;
        end loop;
        for I in 1..N loop
            Prod_Escalar:=Prod_Escalar + V1(I)*V2(I);
        end loop;
        Put("El producto escalar es : ");
        Put(Prod_Escalar);
        New_Line;
    end;
end Producto_Irrestringido;
```

Registros

UNIVERSIDAD DE CANTABRIA

Permiten almacenar datos de diferentes tipos

Se identifican por el nombre del registro y unos campos

Declaración:

```
type Tipo_Escuela is (Teleco, Ciencias, Caminos);

type Alumno is record
   nombre        : String(1..20);
   n_nombre      : Integer range 0..20;
   num_Telefono  : String (1..9);
   Escuela       : Tipo_Escuela := Teleco;
end record;
```

Los campos pueden tener valor inicial

- que será luego asignado a cada variable que se cree de ese tipo

Uso de registros

UNIVERSIDAD DE CANTABRIA

- Completos: por su nombre
- Por componentes: `nombre.campo`

Ejemplos:

```
A1,A2 : Alumno;
...
A2.Nombre := "Pepe                ";  -- 20 caracteres
A1 := A2;
A2.Escuela := Ciencias;
```

Literales de registro

```
A1 := ("Pedro               ",5,"942201020",Caminos);
A2 := (Nombre   => "Juan                ",
       N_Nombre => 4,
       Telefono => "942333333",
       Escuela  => Teleco);
```

Subprogramas y paso de parámetros

UNIVERSIDAD DE CANTABRIA

Los subprogramas encapsulan

- un conjunto de instrucciones
- declaraciones de datos que esas instrucciones necesitan durante su ejecución.

Funcionamiento

- las instrucciones se ejecutan al invocar el subprograma desde otra parte del programa
- se puede hacer intercambio de datos (parámetros)
- al finalizar el subprograma, la ejecución continúa por la instrucción siguiente a la invocación
- las declaraciones de un subprograma se destruyen a su finalización

Subprogramas (cont.)

UNIVERSIDAD DE CANTABRIA

Ventajas

- evitan la duplicidad de código
- son el primer pilar de la división del programa en partes

Pero no sirven para hacer módulos de programa independientes

Un módulo de programa independiente tiene

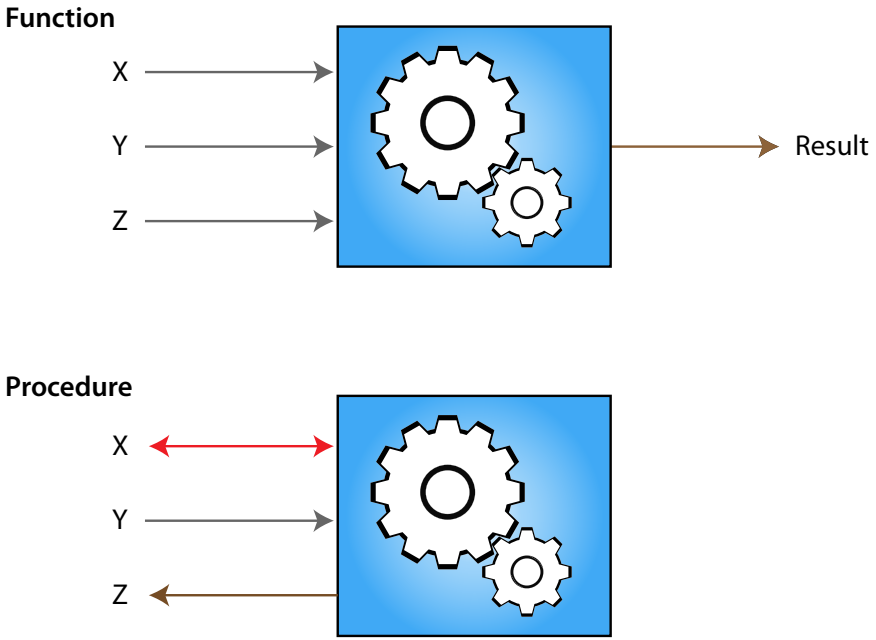
- datos cuya vida es de un ámbito mayor que el del subprograma
- operaciones para manejar esos datos, en forma de subprogramas

Subprogramas (cont.)

UNIVERSIDAD DE CANTABRIA

En Ada hay dos tipos de subprogramas:

- ***funciones***: retornan un dato utilizable en una expresión
- ***procedimientos***: se invocan como una instrucción aparte
  - pueden retornar varios datos, mediante parámetros



Procedimientos

UNIVERSIDAD DE CANTABRIA

Componentes de un procedimiento:

- *nombre*
- *parámetros formales*: datos que intercambia con la parte del programa que lo invoca
  - de entrada (`in`): tipo por omisión
  - de salida (`out`)
  - de entrada y salida (`in out`)
- *declaraciones*
- *instrucciones*

Se pueden compilar aparte (capítulo siguiente) o poner como declaraciones

Estructura de la declaración de un procedimiento

UNIVERSIDAD DE CANTABRIA

```
procedure Nombre (arg1      : in tipo1;
                  arg2      : out tipo2;
                  arg3      : in out tipo3;
                  arg4,arg5 : in tipo4) is
  declaraciones;
begin
  instrucciones;
end Nombre;
```

Los parámetros formales:

- existen sólo dentro del procedimiento
- si son *in*, se tratan como constantes

Ejemplo

UNIVERSIDAD DE CANTABRIA

Dar la vuelta al contenido de un String

```
procedure String_Rev(S: in out String) is
   Lower, Higher : Integer;
   Tmp           : Character;  -- Exchange temp.
begin
   Higher := S'Last;
   for Lower in S'First .. (S'First + S'Last) / 2 loop
      Tmp := S(Lower);         -- Swap
      S(Lower) := S(Higher);
      S(Higher) := Tmp;
      Higher := Higher - 1;    -- Next pair
   end loop;
end String_Rev;
```

Uso de un procedimiento

UNIVERSIDAD DE CANTABRIA

Llamada a un procedimiento

- Se escribe como una instrucción:

  Nombre (parámetros_actuales);

- Cada parámetro actual se asigna a un parámetro formal
  - por orden:

    ```
    Suma (3.0, A, B);
    ```

  - por nombre:

    ```
    Suma (X =>3.0, Y=>A, Suma =>B);
    ```

- Deben respetar las reglas de compatibilidad de tipos
- Los `in` son parámetros de sólo lectura
- Los `out` e `in out` deben ser variables (lectura/escritura)

Uso de un procedimiento (cont.)

UNIVERSIDAD DE CANTABRIA

Es posible dar valor por omisión a un parámetro formal. En ese caso, se puede omitir en la llamada.

```
procedure Desplazamiento_Muelle
   (F : Fuerza; K : Constante_Muelle; T : Temperatura:=25.0)
is ...

Desplazamiento_Muelle (F,K,T);
Desplazamiento_Muelle (F,K);      -- T vale 25.0
```

Ejemplo: producto escalar de dos vectores con procedimientos

Ejemplo de manejo de arrays con un procedimiento

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_IO,Ada.Integer_Text_IO,Ada.Float_Text_IO;
use Ada.Text_IO;
use Ada.Integer_Text_IO;
use Ada.Float_Text_IO;

procedure Producto_Con_Proc is

   Dimension_Max : constant Integer:= 100;
   subtype Indice is Integer range 1..Dimension_Max;
   type Vector is array (Indice) of Float;
   V1,V2         : Vector;
   Dimension     : Indice;
   Prod_Escalar  : Float:=0.0;
```

Ejemplo de manejo de arrays con un procedimiento (cont.)

UNIVERSIDAD DE CANTABRIA

```
procedure Lee_Vector (
      N : in Indice;
      V : out Vector) is
begin
   for I in 1..N loop
      Put("Introduce comp. ");
      Put(I);
      Put(": ");
      Get(V(I));
      Skip_Line;
   end loop;
end Lee_Vector;
```

Ejemplo de manejo de arrays con un procedimiento (cont.)

UNIVERSIDAD DE CANTABRIA

```
begin
   Put("Introduce dimension : ");
   Get(Dimension);
   Skip_Line;
   Lee_Vector(Dimension,V1);
   Lee_Vector(Dimension,V2);

   for I in 1..Dimension loop
      Prod_Escalar:=
      Prod_Escalar + V1(I)*V2(I);
   end loop;

   Put("El producto escalar es : ");
   Put(Prod_Escalar);
   New_Line;
end Producto_Con_Proc;
```

Funciones

UNIVERSIDAD DE CANTABRIA

Son iguales a los procedimientos, pero retornan un valor que se puede utilizar en una expresión

Sólo admiten parámetros de entrada. Declaración:

```
function Nombre (parámetros_formales) return Tipo is
   declaraciones;
begin
   instrucciones;
end Nombre;
```

Al menos una de las instrucciones debe ser

```
return valor;
```

Esta instrucción finaliza la función. Es un error acabar la función sin ejecutarla.

Ejemplo

```
function Cuadrado (X : Float) return Float is
begin
   return X*X;
end Cuadrado;
```

Para invocar la función se hace en una expresión:

```
Y := Cuadrado(Z)*2.0;
```

Ejemplo de una función operador

UNIVERSIDAD DE CANTABRIA

Se pueden definir operadores en Ada mediante funciones cuyo nombre es "operador"

```
type Vector is array (Integer range <>) of Float;

function "+" (A,B : in Vector) return Vector is

   Resultado : Vector(A'range);

begin
   for I in A'range loop
      Resultado(I):=A(I)+B(I);
   end loop;
   return Resultado;
end "+";
```

Ejemplo de uso de esta función

UNIVERSIDAD DE CANTABRIA

```
declare

   N          : Integer := 33;
   V1,V2,V3   : Vector(1..N);

begin
   ...
   V3 := V1+V2;
   ...
end;
```

Reglas de visibilidad

UNIVERSIDAD DE CANTABRIA

Dan respuesta a la pregunta: ¿Desde qué parte del programa es visible (se puede utilizar) una declaración?

Se definen en función de bloques: elementos de programa con la siguiente estructura:

```
encabezamiento
  declaraciones;
begin
  instrucciones;
end;
```

Por ejemplo: procedimientos, funciones, instrucción declare (también paquetes y tareas)

Reglas de visibilidad resumidas

UNIVERSIDAD DE CANTABRIA

Las declaraciones:

- *Regla 1*: Son visibles desde donde han sido definidos hasta el final del bloque
- *Regla 2*: Son visibles dentro del bloque donde se han definido y en los bloques contenidos en él (siempre que se cumpla la regla 1)
- *Regla 3*: Un nombre interno enmascara uno externo
  - excepto en los subprogramas "sobrecargados" (del mismo nombre, pero diferentes parámetros formales)

Ejemplo

Objeto	Visibilidad
P1	1-22
A	2-22
B	2-11, 16-22
P2	3-22
F,G	8-22
P3	9-22
H,I	10-18
P4	11-18
J,K,B	12-15
L,M	19-22

```
1   procedure P1 is
2      A,B : Integer;
3      procedure P2 is
4         D,E : Integer
5      begin
6         ...
7      end P2;
8      F,G : Integer;
9      procedure P3 is
10        H,I : Integer;
11        procedure P4 is
12           J,K,B : Integer;
13        begin
14           ...
15        end P4;
16     begin
17        ...
18     end P3;
19     L,M : Integer;
20  begin
21     ...
22  end P1;
```

Intercambio de información entre subprogramas

UNIVERSIDAD DE CANTABRIA

Podemos intercambiar información de dos maneras:

- *Variables globales*: visibles por varios subprogramas (frente a variables locales, declaradas y visibles sólo localmente por un subprograma)
- *Parámetros*

El método recomendable es el de parámetros

- el uso de variables globales crea dependencias entre partes del programa
- hace más difícil entender lo que hace un procedimiento

Recomendación: salvo que haya muy pocas (una) y esté muy justificado, no usar variables globales.

Bibliografía

[1] J. BARNES. "Programming in Ada 2005", first edition. Addison-Wesley, 2006.

[2] J. BARNES. "Programming In Ada 95", first edition. Addison-Wesley, 1995.

[3] S. Tucker Taft, Robert A. Duff, Randall L. Brukardt, Erhard Ploedereder, and Pascal Leroy (Eds.). “Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1”. LNCS 4348, Springer, 2006.

[4] Productos libres de AdaCore: https://libre.adacore.com/