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

2.1. Introducción

UNIVERSIDAD DE CANTABRIA

Motivación para el lenguaje: "Crisis del software" de mediados de los años 70

- gran número de lenguajes
- necesidad de introducir técnicas de ingeniería de software
  - fiabilidad
  - modularidad
  - programación orientada a objetos (comenzaba en ese momento)

Principios de diseño del lenguaje Ada 83

UNIVERSIDAD DE CANTABRIA

- Fiabilidad
  - legibilidad
  - tipificación estricta
  - excepciones
- Modularidad
- Abstracción de datos y tipos
- Compilación separada
- Concurrencia
- Tiempo Real
- Estandarizado

Notas:

UNIVERSIDAD DE CANTABRIA

- *Legibilidad*: Facilidad para entender el código al leerlo; esto fomenta la fiabilidad porque al entenderse bien el código se cometen menos errores al escribirlo o modificarlo.
- *Tipificación estricta*: El lenguaje obliga a no mezclar datos de distintos tipos en expresiones, y así se detectan más errores de manera automática.
- *Excepciones*: Mecanismo de tratamiento de errores que dificulta que éstos pasen inadvertidos.
- *Modularidad*: Capacidad para partir el programa en módulos o fragmentos independientes.
- *Abstracción de tipos*: Capacidad de hacer módulos independientes de un determinado tipo de datos.
- *Abstracción de datos*: Capacidad de hacer módulos con datos configurables.
- *Compilación separada*: Capacidad de compilar por separado los diferentes módulos del programa.
- *Concurrencia*: Capacidad para ejecutar a la vez varias partes del programa, que atienden de manera simultánea a varios subsistemas o funcionalidades.
- *Tiempo real*: Capacidad para garantizar que los tiempos de respuesta del programa se adaptan a la evolución del entorno exterior.
- *Estandarización*: Normativa clara y aceptada por consenso sobre los elementos que debe ofrecer el lenguaje. La estandarización facilita que las aplicaciones se puedan portar de unos compiladores a otros sin cambios.

Elementos introducidos por Ada 95

UNIVERSIDAD DE CANTABRIA

- Mejor soporte a la programación orientada a objetos
  - extensión de objetos
- Más eficiencia en concurrencia
- Mejor soporte de sistemas de tiempo real

Ada es un lenguaje que incorpora muchas recomendaciones que provienen de la ingeniería de software

- recomendable para grandes proyectos de software

Los principios del Ada son recomendables para todos los desarrollos software, incluso los realizados en otros lenguajes

2.2 Estructura de un programa

UNIVERSIDAD DE CANTABRIA

```
with Nombres_de_Otros_Modulos_de_Programa;
procedure Nombre_Programa is
   declaraciones;
begin
   instrucciones;
end Nombre_programa;
```

Las declaraciones son:

- de *datos*: constantes, variables, tipos
- de *fragmentos de programa*: procedimientos, funciones, paquetes

Notas:

UNIVERSIDAD DE CANTABRIA

Un programa es un procedimiento (procedure) que a su vez puede invocar a otros.

- En C y Java es necesario que el programa sea una función o método llamado **`main`**
- En Java el procedimiento principal que forma un programa puede llamarse como se quiera

La instrucción **`with`** permite indicar nombres de módulos de programas que se van a usar.

El procedimiento tiene un encabezamiento, que contiene las palabras **`procedure`** e **`is`**, y el nombre del procedimiento en medio. Este nombre lo elegimos nosotros.

Después del encabezamiento se ponen las declaraciones y las instrucciones. Éstas últimas se encierran entre **`begin`** y **`end`**.

Las declaraciones crean elementos que se usarán desde las instrucciones.

Las instrucciones se ejecutan en el orden que se escriben, al invocar el procedimiento.

Ejemplo:

UNIVERSIDAD DE CANTABRIA

Programa que pone un mensaje en la pantalla:

```
with Ada.Text_IO;
procedure Sencillo is
   -- sin declaraciones
begin
   Ada.Text_IO.Put("Esto es un programa sencillo");
end Sencillo;
```

Comentarios sobre el ejemplo

UNIVERSIDAD DE CANTABRIA

- Concepto de sangrado:
  - legibilidad
  - muestra visualmente la estructura del código
- Comentarios: comienzan por `--` y acaban al final de la línea
- Las instrucciones y declaraciones acaban en "`;`". El programa también
- Los nombres de identificadores:
  - comienzan por letra
  - siguen letras, números y "`_`"
  - no se distinguen mayúsculas de minúsculas
  - no es aconsejable usar acentos y *ñ*, por los problemas de compatibilidad al pasar a otros sistemas operativos
  - hay un conjunto de nombres reservados

Notas:

UNIVERSIDAD DE CANTABRIA

El *sangrado* es muy importante para facilitar la comprensión del código. Ello facilita cometer menos errores y contribuye por tanto a la fiabilidad.

Recomendación: Utilizar desde el principio los sangrados correctos.

Observar en este ejemplo que *declaración-begin-end* van alineados. Las declaraciones e instrucciones, que están contenidas en el procedimiento, se sitúan alineadas pero más a la derecha.

El *estilo* habitual para los nombres de cosas en Ada es:

- primera letra de cada palabra en mayúsculas, resto en minúsculas
- separar cada palabra del nombre por el carácter '_'
- usar nombres significativos (ej: `Temperatura`, `Presion`, en vez de `a3`, `c5`)

La instrucción `Ada.Text_IO.Put_Line` es equivalente al `System.out.println()` de Java, y similar, aunque menos flexible, al `printf()` de C/C++

Programa sencillo con ventanas

UNIVERSIDAD DE CANTABRIA

```
with Message_Windows;
use Message_Windows;

procedure Sencillo is
   Message :  Message_Window_Type;
begin
   Message:=Message_Window("Esto es un programa sencillo");
   Wait(Message);
end Sencillo;
```

Message_Windows **es un módulo software perteneciente a Win_IO:**

- **No es estándar de Ada**
- **http://www.ctr.unican.es/win_io/**

Notas:

UNIVERSIDAD DE CANTABRIA

La cláusula `use` permite omitir el nombre del módulo en adelante; es opcional

`Message : Message_Window_Type` permite crear una variable llamada `Message`, que representa un dato con información, del tipo `Message_Window_Type`. Este dato contiene la información y operaciones necesarias para pintar una ventana con texto en la pantalla.

En la instrucción `Message:=Message_Window("Esto es un programa sencillo")`
`Message_Window` es la llamada a un procedimiento que crea la ventana y le pone su texto. La ventana se almacena en la variable `Message`.

`Wait(Message)` es la llamada a un procedimiento que espera hasta que el usuario confirma que ha leído la ventana guardada en `Message` pulsando su botón "OK".

2.3. Variables, constantes, y tipos simples

UNIVERSIDAD DE CANTABRIA

La información se guarda en casillas de memoria, que son:

- *variables*: el contenido puede variar
  - algunas tienen nombre
  - otras se crean dinámicamente sin nombre (se verán más adelante)
- *constantes*: el contenido no puede variar
  - las hay con nombre
  - y sin nombre (*literales*): se pone directamente el valor

Todos los datos tienen siempre un tipo asociado:

- tipos *predefinidos*
- tipos *definidos por el usuario*

2.3.1. Tipos predefinidos

UNIVERSIDAD DE CANTABRIA

Son:

Tipo	Valores
`Integer`	entero de 16 bits mínimo [-2**15..2**15-1]
`Float`	real de unos 6 dígitos como mínimo
`Character`	caracter de 8 bits (`Wide_Character`, caracteres internacionales de 16 bits)
`String (1..n)`	texto, definido como una secuencia de `n` caracteres
`Boolean`	Valor lógico: `True` o `False`
`Duration`	Tiempo: número real en segundos

Veremos más adelante que se pueden crear otros tipos, incluidos enteros y reales

Notas:

UNIVERSIDAD DE CANTABRIA

El *bit* es la unidad de almacenamiento de información digital, y representa una cifra binaria (0 ó 1). Poniendo muchos bits juntos podemos hacer muchas combinaciones de valores y representar números grandes.

La notación 2**15 significa 2^{15}

Se distinguen números *enteros* y *reales* porque cada uno tiene sus ventajas e inconvenientes:

- *enteros*: aritmética exacta, rango limitado
- *reales*: aritmética inexacta (sufren errores de redondeo), pero su rango de valores es mucho mayor

Los *textos* se forman con secuencias de muchos caracteres, que a su vez son valores del tipo `Character`

El tipo `String` representa un texto. Su tamaño se decide al ejecutar el programa (según el número de caracteres que tenga) pero una vez definido no puede cambiar

- Veremos más adelante textos de tamaño variable

Los *booleanos* representan valores lógicos verdadero o falso, y se utilizan mucho para tomar decisiones en las instrucciones condicionales.

El tipo `Duration` se usa para representar intervalos relativos de tiempo, en segundos

Atributos de los tipos predefinidos:

UNIVERSIDAD DE CANTABRIA

Tipo	Atributo	Descripción

enteros, reales y enumerados	`tipo'First`
	`tipo'Last`
	`tipo'Image(valor)`
	`tipo'Value(string)`
		`Primer valor`
		`Último valor`
		`Conversión a String`
		`Conversión String a valor numérico o enumerado`

reales	`tipo'Digits` `tipo'Small`	`Número de dígitos` `Menor valor positivo`
discretos (enteros, caracteres y enumerados)	`tipo'Pos(valor)` `tipo'Val(numero)`	`Código numérico de un valor` `Conversión de código a valor`
strings	`s'Length (s es un string concreto)`	`Número de caracteres del string`

Notas:

UNIVERSIDAD DE CANTABRIA

Los *atributos* son elementos del lenguaje que permiten obtener información sobre otro elemento, generalmente un tipo, o un dato.

Los datos *enumerados* son datos cuyos valores son palabras o nombres; los veremos más adelante.

Usaremos con frecuencia el atributo `Image`, que permite la conversión a texto de un dato numérico o enumerado, para poner resultados en la pantalla, ya que en ésta es más fácil escribir texto.

El atributo `Pos` permite obtener el código binario con el que el computador representa los caracteres o los enumerados.

Componentes de los strings

UNIVERSIDAD DE CANTABRIA

Caracteres individuales

```
s(i)
```

Rodajas de string (también son strings)

```
s(i..j)
```

2.3.2. Constantes sin nombre o literales

UNIVERSIDAD DE CANTABRIA

Números enteros

```
13   0   -12   1E7    13_842_234
16#3F8#
```

Números reales

```
13.0    0.0    -12.0    1.0E7
```

Caracteres

```
'a'    'Z'
```

Strings

```
"texto"
```

Booleanos

```
True    False
```

Notas:

UNIVERSIDAD DE CANTABRIA

Los *literales* expresan un valor de un tipo determinado

En los números, los caracteres '_' se ignoran; se usan como *separadores* de millar

Enteros: se pone el valor. La notación `1E7` significa $1 \cdot 10^7$. La notación `16#3F8#` significa el número `3F8` expresado en base `16`.

Reales: Se distinguen por la presencia de la parte fraccionaria separada por el '`.`' decimal. Es obligatorio ponerla aunque sea cero.

Caracteres: se pone el carácter entre apóstrofes o comillas simples

Strings: se pone el texto entre comillas dobles

Booleanos: se pone el valor lógico (en inglés)

2.3.3. Variables y constantes con nombre

UNIVERSIDAD DE CANTABRIA

Características

- Ocupan un lugar en la memoria
- tienen un tipo
- tienen un nombre : identificador
- las constantes no pueden cambiar de valor

Declaración de variables

```
identificador : tipo;
identificador : tipo:=valor_inicial;
```

Declaración de constantes:

```
identificador : constant := valor_inicial;
identificador : constant tipo:=valor_inicial;
```

Ejemplos:

```
Numero_De_Cosas : Integer;
Temperatura     : Float:=37.0;
Direccion       : String(1..30);
Esta_Activo     : Boolean;
Simbolo         : Character:='a';
A,B,C           : Integer;
Max_Num         : constant Integer:=500;
Pi              : constant:=3.1416;
```

Notas:

UNIVERSIDAD DE CANTABRIA

Puede observarse que las declaraciones van al revés que en Java/C: primero se pone la variable y luego el tipo

Se puede elegir dar *valor inicial* o no a una variable

- Si no sabemos qué valor debe tener, es mejor no asignarlo porque así el compilador puede avisarnos si nos equivocamos al usar la variable antes de darle valor

Para las *constantes* es obligatorio dar valor inicial. Si no ponemos el tipo, éste se infiere a partir del literal

El String `Direccion` tiene exactamente 30 caracteres, numerados de 1 a 30. Su tamaño es fijo.

En los *booleanos* es importante que el nombre indique lo que significa que la variable sea verdadera o falsa

- `Esta_Activo` es un buen ejemplo
- `Estado` es un mal nombre: no nos indica qué significa valer true o false

Se pueden crear *varias variables* de golpe, separándolas por comas, como las variables `A,B,C` del ejemplo

Ejemplo de un programa con objetos de datos:

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_Io;

procedure Nombre is

   Tu_Nombre,Tu_Padre : String (1..20);
   N_Nombre,N_Padre   : Integer;
   -- Ejemplo de uso de strings de tamaño variable

begin
   Ada.Text_Io.Put("Cual es tu nombre?: ");
   Ada.Text_Io.Get_Line(Tu_Nombre,N_Nombre);
   Ada.Text_Io.Put("Como se llama tu padre?: ");
   Ada.Text_Io.Get_Line(Tu_Padre,N_Padre);
   Ada.Text_Io.Put_Line("El padre de "&
                        Tu_Nombre(1..N_Nombre)&
                        " es "&Tu_Padre(1..N_Padre));
end Nombre;
```

Notas:

UNIVERSIDAD DE CANTABRIA

En este ejemplo se crean cuatro variables: Dos son Strings y dos son enteros

El ejemplo muestra cómo manejar *strings variables*: se crea un string grande y se usa sólo una parte

- una variable entera guarda el número de caracteres válidos

Las operaciones de I/O usadas son:

- `Put`: Pone un texto en pantalla
- `Put_Line`: es igual que `Put`, pero añade al final del texto un salto de línea.
- `Get_Line`: Lee un string variable del teclado. Entre paréntesis se le pasan dos variables: un string y un entero. El string leído es una secuencia de caracteres finalizada con un salto de línea. Los caracteres se guardan en las primeras posiciones de la variable de tipo string. El salto de línea se lee, pero no se incluye en la variable. El número de caracteres leídos se guarda en el entero.

Para usar la parte válida del string usamos una rodaja de string: `s(1..N)`

Usamos el operador de concatenación: `&`. Sirve para unir dos strings poniendo uno a continuación del otro y formando uno solo. Es necesario, pues la operación `Put` o `Put_Line` sólo admite entre paréntesis un único string.

El mismo ejemplo, con cláusula "use"

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_Io;
use Ada.Text_Io;
procedure Nombre_Con_Use is

   Tu_Nombre,Tu_Padre : String (1..20);
   N_Nombre,N_Padre   : Integer;
   -- Ejemplo de uso de strings de tamaño variable

begin
   Put("Cual es tu nombre?: ");
   Get_Line(Tu_Nombre,N_Nombre);
   Put("Como se llama tu padre?: ");
   Get_Line(Tu_Padre,N_Padre);
   Put_Line("El padre de "&Tu_Nombre(1..N_Nombre)&
            " es "&Tu_Padre(1..N_Padre));
end Nombre_Con_Use;
```

A observar

UNIVERSIDAD DE CANTABRIA

- Para strings de longitud variable
  - Usamos un string grande y de él sólo una parte
  - una variable entera nos dice cuántos caracteres útiles hay
- Uso de rodajas de string
  - para coger la parte útil del string
- Text_IO: `Put`, `Get_Line`, `Put_Line`, `New_Line`
- Cláusula `use`
- Uso de variables

El mismo ejemplo, con ventanas

UNIVERSIDAD DE CANTABRIA

```
with Input_Windows;
use Input_Windows;

procedure Nombres is

   Tu_Nombre,Tu_Padre : String (1..20);
   N_Nombre,N_Padre   : Integer;
   Entrada : Input_Window_Type:=Input_Window("Nombres");

begin
   Create_Entry(Entrada,"Cual es tu nombre?: ","");
   Create_Entry(Entrada,"Como se llama tu padre?: ","");
   Wait(Entrada,"Teclea datos y pulsa OK");
   Get_Line(Entrada,"Cual es tu nombre?: ",Tu_Nombre,N_Nombre);
   Get_Line(Entrada,"Como se llama tu padre?: ",Tu_Padre,N_Padre);
   Put_Line(Entrada,"El padre de "&Tu_Nombre(1..N_Nombre)&
            " es "&Tu_Padre(1..N_Padre));
   Wait(Entrada,"");
end Nombres;
```

Notas:

UNIVERSIDAD DE CANTABRIA

En este ejemplo usamos una ventana de la librería *Win_IO* del tipo **`Input_Window_Type`**, que permite leer datos de teclado.

Cada dato se lee de una *entrada* ("entry") que hay que crear con **`Create_Entry`**. Se identifica mediante una etiqueta (un string) que se pasa como parámetro

Para leer un texto de una entrada se usa **`Get_Line`**, que funciona de modo análogo al **`Get_Line`** de **`Ada.Text_IO`**

Se puede escribir en una zona de la ventana **`Input_Window_Type`** mediante **`Put_Line`**, que funciona también de modo análogo al de **`Ada.Text_IO`**.

Para esperar a que el usuario teclee o pueda ver el resultado hay que invocar a **`Wait`**. En esta llamada se pasa un texto que se muestra también en la ventana.

2.4. Expresiones

UNIVERSIDAD DE CANTABRIA

Permiten transformar datos para obtener un resultado

Se construyen con

- operadores
  - dependen del tipo de dato
  - se pueden definir por el usuario
  - los hay binarios (dos operandos) y unarios (un operando)
- operandos
  - variables
  - constantes
  - funciones

Operadores aritméticos

UNIVERSIDAD DE CANTABRIA

Operan con datos numéricos de un tipo concreto y obtienen datos del mismo tipo.

Operador	Tipo	Operación	Operandos(s)	Resultado
+ -	binario binario	suma resta	numérico numérico	el mismo el mismo
+ -	unario unario	identidad negación	numérico numérico	el mismo el mismo
* / mod rem	binario binario binario binario	multiplicación división módulo resto	numérico numérico Entero Entero	el mismo el mismo Entero Entero
** ** abs	binario binario unario	elevar a elevar a valor absoluto	Entero, Entero >0 Real, Entero numérico	Entero Real el mismo

Notas:

UNIVERSIDAD DE CANTABRIA

El operador de elevar a, **, con datos de tipo real está definido en otro módulo que veremos más adelante

El operador unario + no hace nada, pero se ofrece por simetría y porque se puede redefinir, como veremos más adelante

Operadores relacionales

UNIVERSIDAD DE CANTABRIA

Permiten comparar dos datos del mismo tipo y dar como resultado un booleano

Operador	Tipo	Operación	Operandos(s)	Resultado
=	binario	igual a	cualquiera no limitado	Boolean
/=	binario	distinto de	cualquiera no limitado	Boolean
<	binario	menor que	escalar o string	Boolean
<=	binario	menor o igual que	escalar o string	Boolean
>	binario	mayor que	escalar o string	Boolean
>=	binario	mayor o igual que	escalar o string	Boolean

Los tipos *escalares* son los discretos y los números reales

Los tipos *discretos* son los enteros, caracteres, booleanos y enumerados

Notas:

UNIVERSIDAD DE CANTABRIA

Observar que el operador de *comparación de igualdad* es =

- esto es diferente de lo usado en los lenguajes C, C++ o Java, que usan ==

Veremos más adelante que la *asignación*, que usa el símbolo := no es un operador en Ada

- en C, C++, y Java la asignación usa el símbolo = y es un operador, lo que quiere decir que se puede usar dentro de expresiones más complejas, incluyendo varias asignaciones en la misma expresión

Operadores lógicos

UNIVERSIDAD DE CANTABRIA

Permiten trabajar con valores lógicos (booleanos) y obtener nuevos valores lógicos

Operador	Tipo	Operación	Operandos(s)	Resultado
and or xor	binario binario binario	y o inclusivo o exclusivo	Boolean Boolean Boolean	Boolean Boolean Boolean
not	unario	negación	Boolean	Boolean

Operadores lógicos (cont.)

UNIVERSIDAD DE CANTABRIA

Tablas de verdad:

Operando Izquierdo	Operando Derecho	And	Or	Xor
`false`	`false`	`false`	`false`	`false`
`false`	`true`	`false`	`true`	`true`
`true`	`false`	`false`	`true`	`true`
`true`	`true`	`true`	`true`	`false`

Operador de concatenación

UNIVERSIDAD DE CANTABRIA

Trabaja con dos strings y produce un nuevo string que contiene los caracteres del primero, y a continuación los del segundo

Operador	Tipo	Operación	Operandos(s)	Resultado
&	binario	concatenación	Strings, o string y carácter	String

Precedencia de los operadores, de menor a mayor

UNIVERSIDAD DE CANTABRIA

La precedencia puede modificarse con el uso de paréntesis.

Operador	Operación	Operandos(s)	Resultado
and or xor	y o inclusivo o exclusivo	Boolean Boolean Boolean	Boolean Boolean Boolean
= /= < <= > >=	igual a distinto de menor que menor o igual que mayor que mayor o igual que	cualquiera no limitado cualquiera no limitado escalar o string escalar o string escalar o string escalar o string	Boolean Boolean Boolean Boolean Boolean Boolean
&	Concatenación	Strings,string y carácter	String
+ -	suma resta	numérico numérico	el mismo el mismo

Precedencia de los operadores, de menor a mayor (cont.)

UNIVERSIDAD DE CANTABRIA

Operador	Operación	Operandos(s)	Resultado
+ -	identidad negación	numérico numérico	el mismo el mismo
* / mod rem	multiplicación división módulo resto	Entero Real Entero Real Entero Entero	Entero Real Entero Real Entero Entero
** ** not abs	exponenciación “ negación valor absoluto	Entero, Entero no negativo Real, Entero Boolean numérico	Entero Real Boolean el mismo

Notas:

UNIVERSIDAD DE CANTABRIA

La precedencia de los operadores nos permite saber qué operaciones se hacen primero y cuáles después, si aparecen varios operadores en la misma expresión.

Por ejemplo:

- 3+x*5+y hace primero el producto de x*5 y luego las sumas

La precedencia puede modificarse con paréntesis

- Por ejemplo (3+x)*(5+y) hace primero lo que está dentro de los paréntesis (las sumas) y luego el producto

Sólo pueden usarse paréntesis (), no corchetes [] ni llaves {}, ni paréntesis angulares <>

Comprobación de inclusión

UNIVERSIDAD DE CANTABRIA

En una expresión se puede usar la comprobación de inclusión para obtener un booleano:

```
valor in rango
valor in tipo
```

Un rango se puede expresar como un intervalo de valores escalares (números, caracteres, enumerados, booleanos):

```
valor1 .. valor2
```

Por ejemplo, para comprobar si `a` pertenece al intervalo [3,14]

```
a in 3..14
```

Ejemplos de expresiones

UNIVERSIDAD DE CANTABRIA

aritméticas

```
X+3.0+Y*12.4
Y**N+8.0
```

relacionales

```
X>3.0
N=28   -- compara N con 28. No confundir con la asignación
```

booleanas

```
X>3.0 and X<8.0 -- true si X es mayor que 3 y menor que 8
```

inclusión

```
A in 1..20    -- true o false según A esté o no entre 1 y 20
```

concatenación

```
Mi_Nombre&" texto añadido" -- el resultado es un nuevo string
```

Notas:

UNIVERSIDAD DE CANTABRIA

Observar que los resultados de las operaciones relacionales son booleanos

Las operaciones relacionales no se pueden agrupar de dos en dos

- Por ejemplo, en matemáticas se puede escribir $1 < a < 3$ para comprobar si $a \in (1, 3)$
- En programación el computador haría primero 1<a, obteniendo un booleano (por ejemplo, `true`); luego compararía `true`<3, lo que no tiene sentido
- Para escribir $a \in (1, 3)$ debemos usar dos expresiones relacionales unidas con un operador lógico: `1<a and a<3`
- Podríamos usar la comprobación de inclusión si los límites del intervalo estuviesen incluidos en él. Así para $a \in [1, 3]$: `a in 1.0..3.0`

Instrucción de asignación

UNIVERSIDAD DE CANTABRIA

La asignación permite dar valor a una variable

- en Ada es una instrucción, de modo que no se pueden mezclar varias asignaciones en la misma expresión
- esto contribuye a la sencillez, y a cometer menos errores

Sintaxis

```
variable := expresión;
```

Ejemplos

```
      I, J : Integer;
      X, Y : Float;
...
      I:=3+J;
      J:=3;
      X:=Y;
```

Notas:

UNIVERSIDAD DE CANTABRIA

Observar que la asignación borra el valor anteriormente guardado en la variable

Observar que la variable cuyo valor se cambia se pone a la izquierda, no a la derecha

Tipificación estricta

UNIVERSIDAD DE CANTABRIA

No podemos mezclar datos de distinto tipo:

```
I, J : Integer;
X, Y : Float;
I+3*J -- expresión entera
I+X   -- expresión ilegal
```

Conversión de tipos: Tipo(valor)

```
I+Integer(X)    -- expresión entera
Float(I)+X      -- expresión real
```

Ojo con las operaciones de división

```
3/10      -- vale cero (división entera)
3.0/10.0  -- vale 0.3
```

Expresiones con mod y rem:

UNIVERSIDAD DE CANTABRIA

Con `mod` el resultado tiene el signo del denominador

Con `rem` el resultado es el resto de la división entera (que redondea por debajo)

Ejemplos:

```
 37 mod  10 =  7
-37 mod  10 =  3
 37 rem  10 =  7
-37 rem  10 = -7
 37 mod -10 = -3
-37 mod -10 = -7
 37 rem -10 =  7
-37 rem -10 = -7
```

Ejemplo de un programa con expresiones

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 Nota_Media is
   Nota1,Nota2,Nota3,Nota_Media : Integer;
begin
   Put("Nota del primer trimestre: ");
   Get(Nota1); Skip_Line;
   Put("Nota del segundo trimestre: ");
   Get(Nota2); Skip_Line;
   Put("Nota del tercer trimestre: ");
   Get(Nota3); Skip_Line;
   Nota_Media := (Nota1+Nota2+Nota3)/3;
   Put("Nota Media : ");
   Put(Nota_Media); New_Line;
   Put("Nota Media (otra) : ");
   Put(Float(Nota1+Nota2+Nota3)/3.0);
   New_Line;
end Nota_Media;
```

Notas:

El programa usa el módulo `Ada.Integer_Text_IO` para I/O de enteros y `Ada.Float_Text_IO` para I/O de números reales

El programa declara cuatro variables enteras: `Nota1, Nota2, Nota3, Nota_Media`

Después lee las tres notas por teclado, usando la operación Get de `Ada.Integer_Text_IO`:

- La operación lee un número entero de una línea tecleada, y lo mete en la variable indicada entre paréntesis
- Es importante después del `Get` invocar a `Skip_Line` para consumir el salto de línea que se usa para separar un dato del siguiente
- Es importante antes del `Get` escribir en la pantalla lo que se espera del usuario, para que éste sepa lo que tiene que hacer

Recomendación: Acordarse de hacer un `Skip_Line` por cada línea leída (no escrita) del teclado. Si nos olvidamos alguno, pueden ocurrir comportamientos extraños.

Notas: (cont.)

UNIVERSIDAD DE CANTABRIA

Una vez leídas las notas se calcula la media entera y se muestra con la operación `Put` de `Ada.Integer_Text_IO`

- después del `Put` se invoca a `New_Line` para escribir en la pantalla un salto de línea

Nota: No confundir `Skip_Line` con `New_Line`. El primero afecta al teclado (e indirectamente a la pantalla pues todo lo que se teclea aparece en ella), y el segundo exclusivamente a la pantalla.

Después se calcula la media real y se pone en pantalla con la operación `Put` de `Ada.Float_Text_IO`

- Observar que para calcular la media real hay que convertir la suma de las notas al tipo `Float`

Mismo ejemplo con el atributo 'IMAGE

UNIVERSIDAD DE CANTABRIA

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

procedure Nota_Media is

   Nota1,Nota2,Nota3,Nota_Media : Integer;

begin
   Put("Nota del primer trimestre: ");
   Get(Nota1); Skip_Line;
   Put("Nota del segundo trimestre: ");
   Get(Nota2); Skip_Line;
   Put("Nota del tercer trimestre: ");
   Get(Nota3); Skip_Line;
   Nota_Media := (Nota1+Nota2+Nota3)/3;
   Put_Line("Nota Media : "&Integer'Image(Nota_Media));
   Put_Line("Nota Media (otra) : "&
            Float'Image(Float(Nota1+Nota2+Nota3)/3.0));
end Nota_Media;
```

Notas:

UNIVERSIDAD DE CANTABRIA

En este caso la escritura en pantalla se hace exclusivamente con la operación `Put_Line` de `Ada.Text_IO`

- se usa el atributo `'Image` para convertir números a texto, y la concatenación de strings para crear un único `String` que se imprime con `Put_Line`

Mismo ejemplo con ventanas

UNIVERSIDAD DE CANTABRIA

```
with Input_Windows, Output_Windows;
use Input_Windows, Output_Windows;

procedure Nota_Media is

   Nota1,Nota2,Nota3,Nota_Media : Integer;
   Entrada : Input_Window_Type;
   Salida : Output_Window_Type;

begin
   -- lectura de datos
   Entrada:=Input_Window("Nota Media");
   Create_Entry(Entrada,"Nota del primer trimestre: ",0);
   Create_Entry(Entrada,"Nota del segundo trimestre: ",0);
   Create_Entry(Entrada,"Nota del tercer trimestre: ",0);
   Wait(Entrada,"Introduce datos");
   Get(Entrada,"Nota del primer trimestre: ",Nota1);
   Get(Entrada,"Nota del segundo trimestre: ",Nota2);
   Get(Entrada,"Nota del tercer trimestre: ",Nota3);
```

Mismo ejemplo con ventanas (cont.)

UNIVERSIDAD DE CANTABRIA

```
   -- escribir resultados
   Salida:=Output_Window("Nota Media");
   Nota_Media := (Nota1+Nota2+Nota3)/3;
   Create_Box(Salida,"Nota Media : ",Nota_Media);
   Create_Box(Salida,"Nota Media (otra) : ",
              Float(Nota1+Nota2+Nota3)/3.0);
   Wait(Salida);
end Nota_Media;
```

Notas:

UNIVERSIDAD DE CANTABRIA

En esta versión del ejemplo usamos una ventana del tipo `Input_Window_Type` para leer de teclado, y otra del tipo `Output_Window_Type` para escribir los resultados

En la ventana del tipo `Input_Window_Type` (ver ejemplo anterior usando esta ventana) creamos tres entradas para leer de ellas las tres notas

- Para leer enteros usamos la operación `Get`, siempre después de haber esperado con `Wait` a que el usuario tenga tiempo de teclear

La ventana del tipo `Output_Window_Type` se crea invocando a `Output_Window` con el título de la ventana

- Para escribir en ella debemos crear "cajas" llamando a `Create_Box`, cada una con una etiqueta de texto y un valor
- Luego hay que llamar a la operación `Wait` para dar oportunidad al usuario de ver los datos.

A destacar

UNIVERSIDAD DE CANTABRIA

Uso de `Put` y `Get` para enteros y reales

Uso del `Skip_Line` detrás de cada "`Get`" en el teclado (pero no detrás de `Get_Line`)

- Aunque a veces no es necesario, otras sí (p.e., cuando después hay un `Get_Line`)
- Es conveniente acostumbrarse a ponerlo siempre

También es cómodo usar el atributo:

- `'Image` al escribir
- `'Value` al leer

2.5. Instrucciones de control

UNIVERSIDAD DE CANTABRIA

Permiten modificar el flujo de ejecución del programa

Son:

- **Instrucciones condicionales**
  - `if` (simple, doble, múltiple)
  - `case`: condición discreta (múltiple)
- **Instrucciones de lazo (`loop`)**
  - `for`: número de veces conocido
  - `while`: condición de salida al principio
  - otras condiciones de salida (`exit`)

2.5.1. Instrucción condicional lógica (if)

UNIVERSIDAD DE CANTABRIA

Forma simple:

```
if exp_logica then
   instrucciones;
end if;
```

Forma doble:

```
if exp_logica then
   instrucciones;
else
   instrucciones;
end if;
```

Notas:

UNIVERSIDAD DE CANTABRIA

Esta instrucción permite tomar *decisiones* en base a un valor booleano

Si el booleano es *cierto* se ejecutan las instrucciones contenidas dentro de la instrucción `if`

Si es *falso*, se ejecutan las instrucciones contenidas en la parte `else`, o nada si ésta no existe

Al acabar, se continúa con la siguiente instrucción puesta a continuación del `if`

Instrucción condicional lógica (cont.)

UNIVERSIDAD DE CANTABRIA

Forma múltiple:

```
if exp_logica then
   instrucciones;
elsif exp_logica then
   instrucciones;
elsif exp_logica then
   instrucciones
...
else
   instrucciones;
end if;
```

Notas:

UNIVERSIDAD DE CANTABRIA

En la forma *múltiple*, la decisión de qué instrucciones ejecutar se basa en múltiples valores booleanos.

- Se evalúan en el orden en que aparecen
- En cuanto *uno de ellos es verdad*, se ejecutan las instrucciones colocadas inmediatamente a continuación y la instrucción `if` termina
- Si *ninguno es verdad* se ejecuta la parte `else`, si existe

Ejemplo: cálculo del máximo de A,B y C:

UNIVERSIDAD DE CANTABRIA

```
if A>B then
   max:=A;
else
   max:=B;
end if;
if max<C then
   max:=C;
end if;
```

Notas:

UNIVERSIDAD DE CANTABRIA

En este ejemplo comparamos dos datos, `A` y `B`, para quedarnos con el mayor de los dos en la variable max. Posteriormente comparamos esta variable con `C` para quedarnos con la más grande.

Nota: Evaluación condicional de expresiones booleanas

UNIVERSIDAD DE CANTABRIA

Las expresiones lógicas normales evalúan las dos partes de la expresión

```
if j>0 and i/j>k then ...
```

Posibilidad de error si j=0. Solución para evaluar la segunda parte sólo si se cumple la primera:

```
if j>0 and then i/j>k then ...
```

Lo mismo se puede hacer con la operación `or`:

```
if j>0 or else abs(j)<3 then ...
```

La evaluación condicional es más eficiente

2.5.2. Instrucción condicional discreta (case)

UNIVERSIDAD DE CANTABRIA

Para decisiones que dependen de una expresión discreta se usa la instrucción `case`

- más elegante
- habitualmente más eficiente

Las expresiones discretas son

- enteros
- caracteres
- booleanos
- enumerados

pero no los números reales.

Instrucción case

UNIVERSIDAD DE CANTABRIA

```
case exp_discreta is
   when valor1 =>
        instrucciones;
   when valor2 =>
        instrucciones;
   when valor3 | valor4 | valor5 =>
        instrucciones;
   when valor6..valor7 =>
        instrucciones;
   when others =>
        instrucciones;
end case;
```

Requisitos:

- La cláusula `others` es opcional, pero si no aparece, es obligatorio cubrir todos los posibles valores
- Si no se desea hacer nada poner la instrucción `null`

Notas:

UNIVERSIDAD DE CANTABRIA

La instrucción **`case`** evalúa la expresión discreta y ejecuta las instrucciones que corresponden a su valor. Es equivalente al **`switch`** de Java/C, pero más flexible

Observar que es posible *agrupar* varios valores, o incluso *rangos* de valores, en el mismo caso

- los casos deben ser disjuntos

Si ninguno de los valores corresponde, se ejecutan las instrucciones en la cláusula **`others`**

- La cláusula **`others`** es opcional, pero si no se pone es obligatorio haber cubierto todos los posibles casos

Las instrucciones de cada caso son obligatorias. Si no se desea hacer nada hay que decirlo explícitamente poniendo una instrucción **`null`**, que no hace nada.

Ejemplo: Poner la nota media con letra:

UNIVERSIDAD DE CANTABRIA

```
Nota_Media : Integer:=...;

case Nota_Media is
   when 0..4   => Put_Line("Suspenso");
   when 5..6   => Put_Line("Aprobado");
   when 7..8   => Put_Line("Notable");
   when 9..10  => Put_Line("Sobresaliente");
   when others => Put_Line("Error");
end case;
```

El mismo ejemplo si la nota es un real:

UNIVERSIDAD DE CANTABRIA

```
Nota_Media : Float:=...;

if Nota_Media<0.0 then
   Put_Line("Error");
elsif Nota_Media<5.0 then
   Put_Line("Suspenso");
elsif Nota_Media<7.0 then
   Put_Line("Aprobado");
elsif Nota_Media<9.0 then
   Put_Line("Notable");
elsif Nota_Media<=10.0 then
   Put_Line("Sobresaliente");
else
   Put_Line("Error");
end if;
```

2.5.3. Instrucciones de lazo o bucle

UNIVERSIDAD DE CANTABRIA

Permiten repetir un conjunto de instrucciones

Hay varias instrucciones de lazo, según el número de veces que se quiera hacer el lazo:

- ***indefinido*: lazo infinito**
- ***número máximo de veces conocido* al ejecutar: lazo con variable de control**
- ***número máximo de veces no conocido*:**
  - **el lazo se hace *0 a más veces*: condición de permanencia al principio**
  - **el lazo se hace *1 o más veces*: condición de salida al final.**

A. Lazo infinito

UNIVERSIDAD DE CANTABRIA

Ejemplo:

```
loop
    Put_Line("No puedo parar");
    -- más instrucciones
end loop;
```

B. Lazo con variable de control (for)

UNIVERSIDAD DE CANTABRIA

Una variable discreta toma todos los valores de un rango:

```
for var in rango loop
   instrucciones;
end loop;
```

El rango se escribe de una de las siguientes maneras:

```
valor_inicial..valor_final
tipo
tipo range valor_inicial..valor_final
```

Comentarios:

- La variable se declara en la instrucción y sólo existe durante el lazo
- Toma los valores del rango: [inicial,final], uno por uno
- No se puede cambiar su valor

B. Lazo con variable de control (cont.)

UNIVERSIDAD DE CANTABRIA

Los valores se pueden recorrer en orden inverso:

```
for i in reverse rango loop
```

Ojo. No es lo mismo:

```
reverse 1..10
10..1            -- rango nulo !
```

Ejemplo: Suma de los 100 primeros números

```
Suma : Integer:=0;

for i in 1..100 loop
   Suma:=Suma+i;
end loop;
```

C. Lazo con condición de permanencia al principio (while)

UNIVERSIDAD DE CANTABRIA

Sintaxis:

```
while exp_logica loop
   instrucciones;
end loop;
```

Las instrucciones se ejecutan si la expresión lógica es `true`

Ejemplo: calcular cuántos números enteros *pares* hay que sumar, empezando en uno, para superar el valor 100.

```
J: Integer:=0;
Suma : Integer:=0;

while Suma<=100 loop
   J:=J+2;
   Suma:=Suma+J;
end loop;
```

D. Lazo con condición de salida en cualquier lugar

UNIVERSIDAD DE CANTABRIA

Instrucciones `exit`:

```
exit;                    -- salir del bucle incondicionalmente
exit when condicion;     -- salir del bucle si se cumple la condicion
```

Ejemplo: Calcular la suma de la serie hasta que el término sumado sea menor que 10^{-6}, partiendo de un valor x=1.2

$$\frac{(x-1)}{x} + \frac{(x-1)}{x^2} + \frac{(x-1)}{x^3} + \dots$$

Ejemplo de lazo con condición de salida al final

UNIVERSIDAD DE CANTABRIA

```
X : Float:=1.2;
Suma : Float:=0.0;      -- elemento neutro de la suma
Termino : Float;        -- no requiere valor inicial
Potencia : Float:=1.0; -- elemento neutro del producto

loop
   Potencia:=Potencia*X; -- es más eficiente "*" que "**"
   Termino:=(X-1.0)/Potencia;
   Suma:=Suma+Termino;
   exit when Termino<1.0E-6;
end loop;
```

Ejemplo con instrucciones condicionales y de lazo:

UNIVERSIDAD DE CANTABRIA

Calcular el máximo de un conjunto de valores reales introducidos por teclado

Pseudocódigo:

```
maximo:=menor valor posible
Leer el número de valores
para i desde 1 hasta n
    leer num
    si num> maximo
        maximo=num
    fin si
fin para
```

Ejemplo con instrucciones condicionales y de lazo

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 Maximo is

   Maximo     : Float :=Float'First;
   X          : Float;
   Num_Veces  : Integer;

   ...
```

Ejemplo con instrucciones condicionales y de lazo (cont.)

UNIVERSIDAD DE CANTABRIA

```
begin
    Put("Numero de valores: ");
    Get(Num_Veces);
    Skip_Line;
    for I in 1..Num_Veces loop
        Put("Valor : ");
        Get(X);
        Skip_Line;
        if X>Maximo then
            Maximo:=X;
        end if;
    end loop;
    Put("El maximo es : ");
    Put(Maximo);
    New_Line;
end Maximo;
```

Mismo ejemplo con ventanas

UNIVERSIDAD DE CANTABRIA

```
with Input_Windows;
use Input_Windows;

procedure Maximo is

   Maximo     : Float :=Float'First;
   X          : Float;
   Num_Veces  : Integer;
   Entrada : Input_Window_Type:=Input_Window("Maximo");

begin
   Create_Entry(Entrada,"Numero de valores: ",0);
   Wait(Entrada,"");
   Get(Entrada,"Numero de valores: ",Num_Veces);
   for I in 1..Num_Veces loop
      Create_Entry(Entrada,"Valor : "&Integer'Image(I),0.0);
   end loop;
   Wait(Entrada,"");
```

Mismo ejemplo con ventanas (cont.)

UNIVERSIDAD DE CANTABRIA

```
   for I in 1..Num_Veces loop
      Get(Entrada,"Valor : "&Integer'Image(I),X);
      if X>Maximo then
         Maximo:=X;
      end if;
   end loop;
   Wait(Entrada,"El maximo es : "&Float'Image(Maximo));
end Maximo;
```

Notas:

UNIVERSIDAD DE CANTABRIA

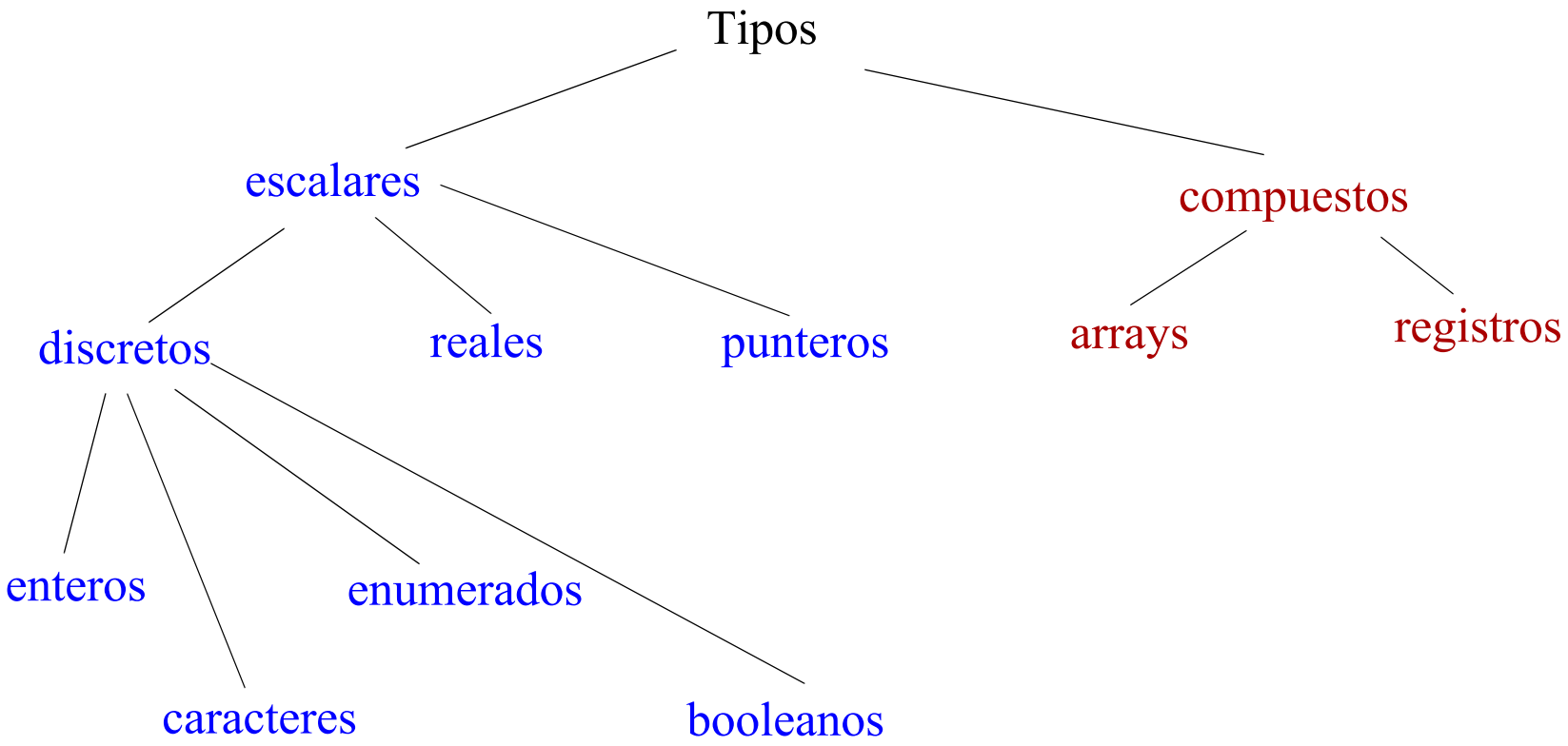
En este ejemplo primero leemos de teclado el número de datos que el usuario desea introducir, y luego creamos tantas entradas como datos se necesitan, para leer un dato de cada una.

Como las entradas deben tener etiquetas distintas, incluimos una parte numérica variable en las etiquetas

2.6. 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 son del mismo tipo

2.6.1. 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;
```

Subtipos anónimos

- es posible crear un subtipo anónimo directamente al crear una variable:

```
I : Integer range 1..100; -- la variable I debe estar entre 1 y 100
                          -- y es del tipo Integer
```

2.6.2. 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;
```

Ejemplo: un tipo equivalente al `double` de Java o C

```
type Real is digits 15;
```

Para mayor eficiencia, sólo conviene crear tipos reales soportados directamente por el hardware

- por ejemplo, el tipo `Real` de arriba

2.6.3. 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_Enumerado);
```

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

Entrada/Salida de tipos escalares (cont.)

UNIVERSIDAD DE CANTABRIA

Podemos poner cláusulas `use` para estos módulos después de crearlos.

```
use Color_IO;
C : Color;
...
Get(C);
Skip_Line;
```

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;
```

Mismo ejemplo con ventanas

UNIVERSIDAD DE CANTABRIA

```
with Input_Windows, Output_Windows;
use Input_Windows, Output_Windows;
procedure Nota_Media_Enum is

    type Nota_Num is range 0..10;
    type Nota_Letra is
       (Suspenso,Aprobado,Notable,Sobresaliente);
    Nota1,Nota2,Nota3,Nota_Media : Nota_Num;
    Nota_Final : Nota_Letra;
    Entrada : Input_Window_Type;
    Salida : Output_Window_Type;

begin
    -- lectura de datos
    Entrada:=Input_Window("Nota Media");
    Create_Entry(Entrada,"Nota del primer trimestre: ",0);
    Create_Entry(Entrada,"Nota del segundo trimestre: ",0);
    Create_Entry(Entrada,"Nota del tercer trimestre: ",0);
    Wait(Entrada,"Introduce datos");
```

Mismo ejemplo con ventanas (cont.)

UNIVERSIDAD DE CANTABRIA

```
   Get(Entrada,"Nota del primer trimestre: ",Integer(Nota1));
   Get(Entrada,"Nota del segundo trimestre: ",Integer(Nota2));
   Get(Entrada,"Nota del tercer trimestre: ",Integer(Nota3));

   -- escribir resultados
   Salida:=Output_Window("Nota Media");
   Nota_Media := (Nota1+Nota2+Nota3)/3;
   Create_Box(Salida,"Nota Media : ",Nota_Num'Image(Nota_Media));
   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;
   Create_Box(Salida,"Nota Final : ",
              Nota_Letra'Image(Nota_Final));
   Wait(Salida);
end Nota_Media_Enum;
```

Notas:

UNIVERSIDAD DE CANTABRIA

La operación `Get` de `Input_Windows` sólo admite datos del tipo `String`, `Integer`, o `Float`

- No admite datos del tipo `Nota`
- Por ello hemos hecho una conversión de las variables `Nota1`, `Nota2`, y `Nota3` al tipo `Integer`
- Observar que esta conversión funciona en ambos sentidos (para obtener el valor de la variable, o para asignarle valor a través de la llamada al procedimiento)

La operación `Create_Box` de `Output_Windows` sólo admite datos del tipo `String`, `Integer`, o `Float`

- Por ello hemos usado el atributo `'Image` para convertir una nota a texto, y luego un enumerado a texto

A destacar

UNIVERSIDAD DE CANTABRIA

Las cláusulas `use` permitirían reducir el texto

Los tipos subrango permiten detectar errores de rango de manera automática

- por ejemplo si ponemos una nota menor que cero o mayor que 10
- más adelante aprenderemos a tratar estos errores

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

- pues ya hemos cubierto todos los casos

2.6.4. 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 del *índice* que debe ser discreto

Declaración:

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

El rango no tiene por qué ser un número (puede ser un carácter o un enumerado) ni necesita empezar en cero

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..4) 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;
```

Notas:

UNIVERSIDAD DE CANTABRIA

Las variables del tipo:

- `Vector_3D` tendrán tres números reales, cuyos índices serán 1,2,3
- `Num_Dias_Mes` tendrán 12 enteros no negativos, con índices de 1 a 12
- `Num_Dias` tendrán 12 enteros no negativos, correspondientes a los meses de `Enero`, `Febrero`, ... , `Diciembre`
- `Matriz` son arrays bidimensionales de 12 números reales. El primer índice (que puede indicar, por ejemplo, las filas) tiene 3 valores: 1,2,3. El segundo índice (que puede indicar, por ejemplo, las columnas) tiene 4 valores: 1,2,3,4
- `Tabla_Temperaturas` tendrán 24x12=288 valores reales del tipo `Temp`, uno por cada combinación hora-mes.
- `Bool` tendrán `N` valores booleanos, siendo `N` un valor conocido en tiempo de ejecución. Eso sí, una vez creado el tipo, su tamaño ya no cambia, aunque `N` cambie.

Uso del array:

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

Ejemplos:

```
V1, V2 : Vector_3D;
M : Matriz;
Dias : Num_Dias;
T : Tabla_Temperaturas;
B : array (1..100) of Float; -- Array de tipo anónimo
Contactos : Bool;
...
V1(1):=3.0+V2(3);
M(2,3):=M(1,1)*2.0 + B(12);
Dias(Enero):=31;
T(20,Febrero):=23.1;
V2:=V1;
V1(1..2):=V2(2..3);
```

Notas:

UNIVERSIDAD DE CANTABRIA

Declaraciones

- `V1` y `V2` son arrays de 3 números reales
- `M` es un array bidimensional de 12 números reales
- `Dias` es un array de 12 enteros no negativos
- `T` es un array de 288 valores reales del tipo `Temp`
- `B` es un array de 100 números reales, numerados de 1..100. Este ejemplo muestra que se puede crear un array con un tipo anónimo, directamente al crear la variable
  - Hay que tener cuidado, pues los tipos anónimos son incompatibles, aunque se definan igual. Por ejemplo si creamos `B2 : array (1..100) of Float`, no podremos mezclar `B` y `B2`
- `Contactos` es un array de `N` valores booleanos

Instrucciones

- Las cuatro primeras instrucciones muestran cómo usar una casilla individual del array
- La instrucción `V2:=V1` copia todas las casillas de `V1` en `V2`
- La instrucción `V1(1..2):=V2(2..3)` copia las casillas 2 a 3 de `V2` en las casillas 1 a 2 de `V1`.
  - El tamaño de ambos fragmentos debe coincidir

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; -- la variable i toma sucesivamente todos
end loop;       -- los valores del índice
```

`Tipo'Length` o `Array'Length`: indica la longitud del array (el número de elementos)

```
for i in 1..V1'Length-1 loop
    V1(i):=0.0; -- la variable i toma sucesivamente todos
end loop;       -- los índices desde el 1 hasta el penúltimo
```

`Tipo'First` o `Array'First`: es el primer índice

`Tipo'Last` o `Array'Last`: es el último índice

Literales de array

UNIVERSIDAD DE CANTABRIA

Permiten expresar arrays a partir de todos los valores

```
V1:=(0.0, 3.2, 1.2);
Dias:=(31,28,31,30,31,30,31,31,30,31,30,31);
```

O agrupándolos por rangos del índice

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

Funciona también con rodajas de arrays

```
B(1..4):=(0.0,1.0,2.0,3.0);
```

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**
  - el tamaño no va en el tipo sino que se expresa al crear la variable

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 v1: 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;
```

Ejemplo versión 2: arrays de tamaño variable

UNIVERSIDAD DE CANTABRIA

Modalidad 2: declarar los arrays del tamaño justo

Usar para ello la instrucción `declare`, que se puede poner junto a otras instrucciones y permite declarar nuevas variables, tipos, ...

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

Las declaraciones hechas en un `declare` sólo se pueden usar hasta el correspondiente `end`

Ejemplo v2: 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 V1'range loop
         Put("Introduce componente ");
         Put(I);
         Put(": ");
```

Ejemplo v2: 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 V1'range 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

El tamaño se deja indeterminado, y se define al declarar la variable

```
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 v3: 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 v3: arrays no restringidos

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 V1'range 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;
```

Mismo ejemplo con ventanas

UNIVERSIDAD DE CANTABRIA

```
with Input_Windows, Output_Windows;
use Input_Windows, Output_Windows;
procedure Producto_Irrestringido is
   type Vector is array (Positive range <>) of Float;
   N             : Positive;
   Prod_Escalar : Float:=0.0;
   Entrada : Input_Window_Type;
   Salida : Output_Window_Type;
begin
   Entrada:=Input_Window("Producto Escalar");
   Create_Entry(Entrada,"Introduce dimension : ",0);
   Wait(Entrada,"");
   Get(Entrada,"Introduce dimension : ",N);
   declare
      V1,V2 : Vector(1..N);
   begin
      -- Crea las entradas
      for I in V1'range loop
         Create_Entry(Entrada,"V("&Integer'Image(I)&")",0.0);
      end loop;
```

Mismo ejemplo con ventanas (cont.)

UNIVERSIDAD DE CANTABRIA

```
      -- lee V1
      Wait(Entrada,"Introduce Vector V1");
      for I in V1'range loop
         Get(Entrada,"V("&Integer'Image(I)&")",V1(I));
      end loop;
      -- lee V2
      Wait(Entrada,"Introduce Vector V2");
      for I in V2'range loop
         Get(Entrada,"V("&Integer'Image(I)&")",V2(I));
      end loop;
      -- calcula resultado
      for I in V1'range loop
         Prod_Escalar:=Prod_Escalar + V1(I)*V2(I);
      end loop;
      -- escribe resultado
      Salida:=Output_Window("Producto Escalar");
      Create_Box(Salida,"El producto escalar es : ",Prod_Escalar);
      Wait(Salida);
   end;
end Producto_Irrestringido;
```

2.6.5. Registros

UNIVERSIDAD DE CANTABRIA

Permiten agrupar datos de diferentes tipos, similarmente a los atributos de un objeto en C++ o Java

- veremos más adelante cómo asociarles métodos

Se identifican por el nombre del registro y unos campos

Declaración:

```
type Nombre_Tipo is record
   nombre_campo1 : tipo_campo1;
   nombre_campo2 : tipo_campo2;
   ...
end record;
```

Registros (cont.)

UNIVERSIDAD DE CANTABRIA

Ejemplo

```
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

- 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);
```

Notas:

UNIVERSIDAD DE CANTABRIA

En los ejemplos anteriores

- A1 y A2 son dos registros del tipo Alumno
- A2.Nombre es el campo Nombre de A2
- Los registros permiten la asignación: se copian todos los campos

En los literales de registro hay dos formatos:

- Poner los valores de los campos, separados por comas (deben ir en el mismo orden)
- Poner delante del valor de cada campo su nombre y una flecha =>; en este formato los campos se pueden dar en el orden que se prefiera

Ejemplo: Diagrama estadístico de barras

UNIVERSIDAD DE CANTABRIA

Se desea hacer un programa que haga un gráfico de barras de varios valores medidos, caracterizados cada uno por un valor máximo y uno mínimo

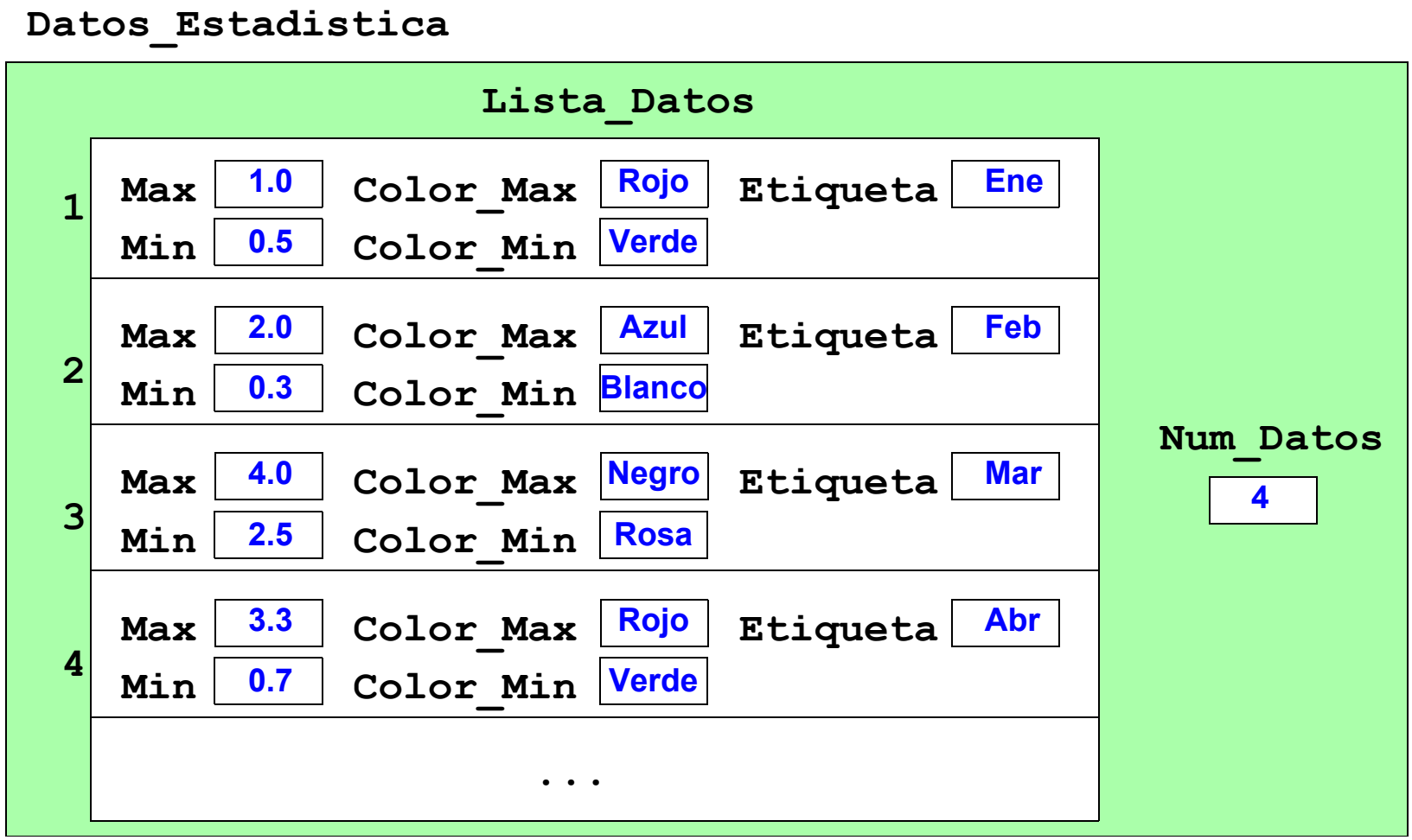
- El número de datos (y de barras) es variable y el máximo es 20

Para cada dato hay que poder expresar el color en que se pinta la barra de máximo y la de mínimo, así como una etiqueta de 3 caracteres

- esta información se guarda en un registro por dato, junto a los valores máximo y mínimo

Los registros se guardan en un array, ocupando sus primeras posiciones. Una variable entera indica cuántos datos hay. Ambos (array y entero) se guardan en otro registro

Ejemplo: Diagrama estadístico de barras (cont.)



Notas:

UNIVERSIDAD DE CANTABRIA

`Datos_Estadística` es un registro con los campos:

- `Lista_Datos`: es un array de registros del tipo Dato, de 20 casillas, numeradas de 1 a 20
- `Num_Datos`: es un entero que indica cuántos datos válidos hay almacenados en el momento actual (desde el principio del array)

`Dato` es un registro con los siguientes campos

- `Max`: valor real que indica el valor máximo
- `Min`: valor real que indica el valor mínimo
- `Color_Max`: color con el que pintar la barra del máximo (un enumerado)
- `Color_Min`: color con el que pintar la barra del máximo (un enumerado)
- `Etiqueta`: texto de 3 caracteres

Ejemplo: Diagrama estadístico de barras (cont.)

UNIVERSIDAD DE CANTABRIA

```
with Graphics_Windows;
use Graphics_Windows;

procedure Estadistica is
   Max_Datos : constant Integer:=20;
   type Colores is (Negro, Blanco, Rojo, Verde, Azul,
                    Gris, Amarillo, Azul_Claro, Rosa);

   type Dato is record
      Max,Min : Float;
      Color_Max,Color_Min : Colores;
      Etiqueta : String(1..3):="   ";
   end record;

   type Lista_Datos is array(1..Max_Datos) of Dato;

   type Datos_Estadistica is record
      Lista : Lista_Datos;
      Num_Datos : Integer range 0..Max_Datos:=0;
   end record;
```

Ejemplo: Diagrama estadístico de barras (cont.)

UNIVERSIDAD DE CANTABRIA

```
   Dat : Datos_Estadistica;
   Grafico : Canvas_Type;
   Ancho, Alto, Pos : Integer;
   Factor, Maximo : Float;
   Esp_Entre_Col : constant Integer:=2;
   Transforma : array(Colores) of Color_Type:=
     (Black, White, Red, Green, Blue, Gray, Yellow, Cyan, Magenta);

begin
   -- Poner datos
   Dat.Num_Datos:=8;

   -- Uno por uno
   Dat.Lista(1).Max:=20.0;
   Dat.Lista(1).Min:=10.0;
   Dat.Lista(1).Color_Max:=Rojo;
   Dat.Lista(1).Color_Min:=Verde;
   Dat.Lista(1).Etiqueta:="AAA";
```

Ejemplo: Diagrama estadístico de barras (cont.)

UNIVERSIDAD DE CANTABRIA

```
-- Usando literales
Dat.Lista(2):=(12.0, 8.0, Azul, Negro, "BBB");
Dat.Lista(3):=(22.0, 18.0, Rojo, Verde, "CCC");
Dat.Lista(4):=(32.0, 28.0, Azul, Negro, "DDD");
Dat.Lista(5):=(17.0, 8.0, Azul, Negro, "EEE");
Dat.Lista(6):=(19.0, 18.0, Rojo, Verde, "FFF");
Dat.Lista(7):=(8.0, 6.0, Azul, Negro, "GGG");
Dat.Lista(8):=(25.0, 16.0, Azul, Negro, "HHH");

Maximo:=32.0;

-- Hacer Dibujo
Grafico:=Canvas(640,480,"Estadistica");
Set_Font_Size(Grafico,12);
Ancho:=(600-Dat.Num_Datos*Esp_Entre_Col)/Dat.Num_Datos;
Pos:=20;
Factor:=400.0/Maximo;
```

Ejemplo: Diagrama estadístico de barras (cont.)

UNIVERSIDAD DE CANTABRIA

```
   for I in 1..Dat.Num_Datos loop
      Alto:=Integer(Factor*Dat.Lista(I).Max);
      Set_Fill(Grafico,Transforma(Dat.Lista(I).Color_Max));
      Draw_Rectangle(Grafico,(Pos,440-Alto),Ancho,Alto);
      Alto:=Integer(Factor*Dat.Lista(I).Min);
      Set_Fill(Grafico,Transforma(Dat.Lista(I).Color_Min));
      Draw_Rectangle(Grafico,(Pos,440-Alto),Ancho,Alto);
      Draw_Text(Grafico,(Pos+2,460),Dat.Lista(I).Etiqueta);
      Pos:=Pos+Ancho+Esp_Entre_Col;
   end loop;
   Wait(Grafico);
end Estadistica;
```

Notas:

UNIVERSIDAD DE CANTABRIA

Creamos un enumerado llamado Colores para nuestros colores, con los valores Negro, Blanco, Rojo, ...

Usaremos un objeto de la clase Canvas_Type, de la librería Graphics_Windows para dibujar el gráfico de barras.

Utilizamos el array Transforma para convertir nuestros colores a los utilizados en Graphics_Windows. Cada casilla del array tiene como índice un enumerado del tipo Colores, y un valor del tipo Color_Type

El programa comienza rellenando la variable Datos_Estadistica con información para 8 datos a dibujar (a modo de prueba)

- Primero se muestra cómo hacerlo rellenando cada campo individualmente, y luego mediante literales

También se crea la variable `Maximo` igual al máximo de todos los valores máximos; se usa para determinar la escala del dibujo

La referencia en el lienzo para las coordenadas (X,Y) son la parte inferior izquierda: punto (0,0)

Notas:

UNIVERSIDAD DE CANTABRIA

Para hacer el dibujo se hacen los siguientes pasos iniciales:

- Se crea el lienzo (del tipo Canvas_Type), que es una ventana para dibujar; hay que dar el tamaño en píxeles y el título
- Se pone el tamaño de la fuente de texto para el lienzo a 12 puntos
- Se inicializan las variables siguientes:
  - ancho (igual al ancho de cada barra, obtenido en función del número de barras que caben en el dibujo)
  - pos: indica la posición X de la parte izquierda de la siguiente barra a dibujar; se empieza en 20 para dejar un poco de margen
  - factor: es el factor de escala del dibujo en vertical, medido en píxeles/unidad de los valores

Luego se comienza un bucle para pintar cada barra

- Se calcula la altura de la barra en píxeles
- Se pone el color del relleno de las figuras al color del valor máximo
- Se dibuja el rectángulo de la barra del máximo
- Luego se hace lo mismo para la barra del mínimo y finalmente se pone la etiqueta de texto debajo de las barras y se aumenta pos para dibujar la siguiente barra en el lugar apropiado

A observar

UNIVERSIDAD DE CANTABRIA

- Uso de un registro que contiene un array de registros
- Uso de la clase `Graphics_Windows`
- Uso de literales de registros
- Uso de un array para transformar datos de un tipo enumerado a otro

2.7. 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 (a través de los 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

Son equivalentes a las funciones de C y similares a los métodos de Java/C++

- aunque no se asocian necesariamente con un objeto como pasa en Java o C++

2.7.1. 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 en la parte declarativa de otro módulo

- por ejemplo dentro de otros subprogramas

Notas:

UNIVERSIDAD DE CANTABRIA

Los parámetros o argumentos de los subprogramas Ada pueden ser de entrada, salida, o entrada/salida

En Ada no se indica si el parámetro se pasa por valor (se pasa una copia) o por referencia (se pasa la dirección de memoria en la que se encuentra); esto lo elige el compilador para conseguir la máxima eficiencia

- si el dato es corto lo pasa por valor
- si es grande lo hace por referencia

Por tanto, en Ada el programador expresa la intención de uso del dato, y el compilador elige la mejor implementación posible

En C/Java la situación es diferente. Los parámetros se pasan sólo por valor.

- En C, si se desea pasar un parámetro por referencia se crea una referencia explícita, con un puntero que se pasa por valor; ello es necesario si se desea que la función modifique el parámetro original
- En Java las variables de los tipos elementales sólo se pueden pasar por valor, pues no hay punteros explícitos; en cambio, los objetos se pasan siempre por referencia (el valor de su dirección)

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;
```

Ejemplo

UNIVERSIDAD DE CANTABRIA

Calcular la suma de dos números y mostrarla en la pantalla

```
procedure Suma (X,Y : in Float; Suma : out Float) is
begin
    Suma:=X+Y;
    Ada.Text_IO.Put_Line("Suma: "&Float'Image(Suma));
end Suma;
```

Los parámetros formales:

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

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 expresiones
- Los `out` e `in out` deben ser variables

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: mostraremos cómo hacer el producto escalar de dos vectores con procedimientos

Ejemplo de manejo de arrays con procedimientos

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;
```

Ejemplo de manejo de arrays con procedimientos (cont.)

UNIVERSIDAD DE CANTABRIA

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

Ejemplo de manejo de arrays con procedimientos (cont.)

UNIVERSIDAD DE CANTABRIA

```
procedure Calcula_Prod_Escalar (
      N    : in  Indice;
      V1   : in  Vector;
      V2   : in  Vector;
      Prod : out Float) is
begin
   Prod:=0.0;
   for I in 1..N loop
      Prod:=Prod+ V1(I)*V2(I);
   end loop;
end Calcula_Prod_Escalar;
```

Ejemplo de manejo de arrays con procedimientos (cont.)

UNIVERSIDAD DE CANTABRIA

```
begin
   Put("Introduce dimension : ");
   Get(Dimension);
   Skip_Line;
   Lee_Vector(Dimension,V1);
   Lee_Vector(Dimension,V2);
   Calcula_Prod_Escalar(Dimension,V1,V2,Prod_Escalar);
   Put("El producto escalar es : ");
   Put(Prod_Escalar);
   New_Line;
end Producto_Con_Proc;
```

Notas:

UNIVERSIDAD DE CANTABRIA

Podemos ver que hemos creado dos procedimientos, dentro de las declaraciones del procedimiento que representa el programa principal

- `Lee_Vector` sirve para leer los componentes de un vector desde teclado; se le pasa como parámetro de entrada la dimensión del vector y como parámetro de salida la variable donde se deposita el vector leído
- `Calcula_Producto_Escalar` tiene como parámetros de entrada la dimensión de los vectores y los dos vectores a multiplicar, y da como parámetro de salida el resultado, que es un número real

El programa principal se encarga de:

- Leer la dimensión del vector
- Invocar a `Lee_Vector` dos veces para leer cada uno de los dos vectores
- Llamar a `Calcula_Producto_Escalar`
- Mostrar en pantalla el resultado

Puede observarse la ventaja del procedimiento `Lee_Vector`, que permite usar dos veces las mismas instrucciones sin necesidad de repetirlas.

2.7.2. 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 ejecutar esta instrucción

Ejemplo:

UNIVERSIDAD DE CANTABRIA

Cálculo del cuadrado de un número real

```
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;

-- Suma de vectores
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

```
declare

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

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

Librería de Servicios Numéricos

UNIVERSIDAD DE CANTABRIA

```
package Ada.Numerics is
pragma Pure (Numerics);

   Argument_Error : exception;

   Pi : constant :=
          3.14159_26535_89793_23846_26433_
            83279_50288_41971_69399_37511;

   e : constant :=
          2.71828_18284_59045_23536_02874_
            71352_66249_77572_47093_69996;

end Ada.Numerics;
```

Notas:

UNIVERSIDAD DE CANTABRIA

La librería `Ada.Numerics` define las constantes elementales `e` y `pi`

En el código de arriba se ha puesto la parte fraccionaria en dos líneas, para que quepa, pero en realidad debe ir en una sola

Librería estándar de funciones matemáticas

UNIVERSIDAD DE CANTABRIA

```
package Ada.Numerics.Elementary_Functions is

   function Sqrt     (X           : Float) return Float;
   function Log      (X           : Float) return Float;
   function Log      (X, Base     : Float) return Float;
   function Exp      (X           : Float) return Float;
   function "**"     (Left, Right : Float) return Float;

   function Sin      (X           : Float) return Float;
   function Sin      (X, Cycle    : Float) return Float;
   function Cos      (X           : Float) return Float;
   function Cos      (X, Cycle    : Float) return Float;
   function Tan      (X           : Float) return Float;
   function Tan      (X, Cycle    : Float) return Float;
   function Cot      (X           : Float) return Float;
   function Cot      (X, Cycle    : Float) return Float;
```

Librería estándar de funciones matemáticas (cont.)

UNIVERSIDAD DE CANTABRIA

```
function Arcsin   (X           : Float) return Float;
function Arcsin   (X, Cycle    : Float) return Float;
function Arccos   (X           : Float) return Float;
function Arccos   (X, Cycle    : Float) return Float;

function Arctan
  (Y : Float; X : Float := 1.0) return Float;

function Arctan
  (Y : Float; X : Float := 1.0; Cycle : Float) return  Float;

function Arccot
  (X : Float; Y : Float := 1.0) return Float;

function Arccot
  (X : Float; Y : Float := 1.0; Cycle : Float) return  Float;
```

Librería estándar de funciones matemáticas (cont.)

UNIVERSIDAD DE CANTABRIA

```
   function Sinh     (X : Float) return Float;
   function Cosh     (X : Float) return Float;
   function Tanh     (X : Float) return Float;
   function Coth     (X : Float) return Float;
   function Arcsinh  (X : Float) return Float;
   function Arccosh  (X : Float) return Float;
   function Arctanh  (X : Float) return Float;
   function Arccoth  (X : Float) return Float;

end Ada.Numerics.Elementary_Functions;
```

Notas:

UNIVERSIDAD DE CANTABRIA

La librería `Ada.Numerics.Elementary_Functions` define las funciones aritméticas elementales:

- `Sqrt(x)`: raíz cuadrada
- `Log(x)`: logaritmo neperiano
- `Log(x,base)`: logaritmo en la base especificada
- `"**"`: elevar un número real a otro
- `Sin(x), Cos(x), Tan(x), Cot(x)`: funciones trigonométricas con radianes: seno, coseno, tangente, cotangente
- `Sin(x,cycle), Cos(x,cycle), Tan(x,cycle), Cot(x,cycle)`: funciones trigonométricas con ángulos en otras unidades. `Cycle` es el número de unidades en la circunferencia; por ejemplo, para grados sexagesimales utilizar `cycle => 360.0`
- `Arcsin(x), Arccos(x)`: funciones trigonométricas inversas en radianes: arcoseno y arcocoseno
- `Arcsin(x,cycle), Arccos(x,cycle)`: funciones trigonométricas inversas con ángulos en otras unidades

Notas:

UNIVERSIDAD DE CANTABRIA

- **`Arctan(y,x), Arccot(y,x)`**: funciones trigonométricas inversas en radianes: arcotangente(y/x) y arcocotangente(y/x); se usan los signos de **`y`** y **`x`** para obtener el ángulo correcto en los cuatro cuadrantes; por ejemplo, para **`arctan`**, si **`y`** es el seno del ángulo y **`x`** el coseno se obtiene el ángulo correctamente; es equivalente al **`atan2()`** de C o Java
- **`Arctan(y,x,cycle), Arccot(y,x,cycle)`**: Igual que las operaciones anteriores, pero con el ángulo en otras unidades
- **`Sinh(x), Cosh(x), Tanh(x), Coth(x)`**: funciones trigonométricas hiperbólicas: seno, coseno, tangente y cotangente hiperbólicas
- **`Arcsinh(x), Arccosh(x), Arctanh(x), Arccoth(x)`**: funciones trigonométricas hiperbólicas inversas

Ejemplos de uso de las funciones matemáticas

UNIVERSIDAD DE CANTABRIA

```
with Ada.Numerics.Elementary_Functions;
use  Ada.Numerics.Elementary_Functions;

procedure Prueba is

   A,B,C : Float;

begin
   A:=Sqrt(2.13);
   B:=Log(A);           -- logaritmo neperiano
   C:=Sin(B);           -- B en radianes
   A:=Sin(B,360.0);     -- B en grados
   B:=A**C;             -- elevar A a C
end Prueba;
```

2.8 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, que veremos más adelante)

Reglas de visibilidad resumidas

UNIVERSIDAD DE CANTABRIA

Las declaraciones:

- *Regla 1*: Son visibles desde donde aparecen hasta el final del bloque
- *Regla 2*: Son visibles dentro del bloque donde aparecen 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 (líneas)
P1	1-22
A	2-22
B (de P1)	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.