Parte I: Programación en 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

Elementos introducidos por Ada 95

UNIVERSIDAD DE CANTABRIA

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

El Ada es el lenguaje 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

Por ello es aconsejable como primer lenguaje de programación.

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

Ejemplo:

UNIVERSIDAD DE CANTABRIA

```
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*
  - hay un conjunto de nombres reservados

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/`

2.3. Variables, constantes, y tipos simples

UC
UNIVERSIDAD DE CANTABRIA

La información se guarda en casillas de memoria

- ***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

UC
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` para 16 bits)
`String (1..n)`	texto o secuencia de caracteres
`Boolean`	`True` o `False`
`Duration`	número real en segundos

Atributos de los tipos predefinidos:

UNIVERSIDAD DE CANTABRIA

Tipo	Atributo	Descripción

enteros, reales y enumerados	`tipo'FIRST`
	`tipo'LAST`
	`tipo'IMAGE(numero)`
	`tipo'VALUE(string)`
		`Primero valor`
		`Ultimo valor`
		`Conversión a String`
		`Conversión String a número`

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`

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
```


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:

UNIVERSIDAD DE CANTABRIA

```
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;
```


© Javier Gutiérrez, Michael González
26/sept/08

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;
```


© Javier Gutiérrez, Michael González
26/sept/08

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

UC
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;
```


2.4. Expresiones

UC
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**
  - **binarios (dos operandos) y unarios (un operando)**
- **operandos**
  - **variables**
  - **constantes**
  - **funciones**

Precedencia de los operadores

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 (cont.)

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

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
```


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;
```

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;
```


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;
```


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;
```

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;
```


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;
```

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**
- **posiblemente 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`

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;
```


© Javier Gutiérrez, Michael González
26/sept/08

2.5.3. Instrucciones de lazo

UNIVERSIDAD DE CANTABRIA

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.

© Javier Gutiérrez, Michael González
26/sept/08

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;
```


© Javier Gutiérrez, Michael González
26/sept/08

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

UNIVERSIDAD DE CANTABRIA

Sintaxis:

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

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;
```


© Javier Gutiérrez, Michael González
26/sept/08

D. Lazo con condición de salida en cualquier lugar

UC UNIVERSIDAD DE CANTABRIA

Instrucciones exit:

```
exit;
exit when 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

UC 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:

```
Leer el número de valores
for i desde 1 hasta n
   leer num
   si num> maximo
       maximo=num
   fin si
fin lazo
```


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.)

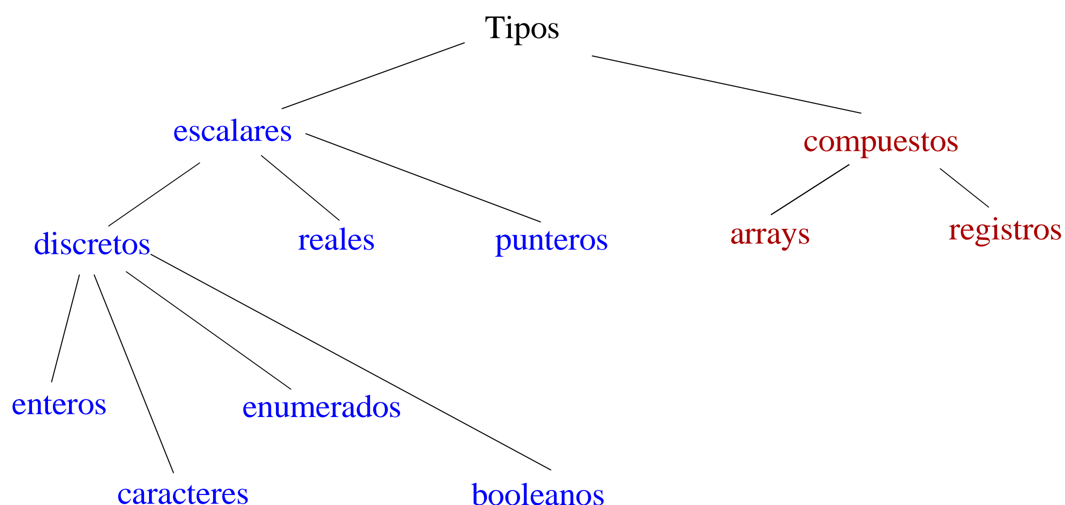
UNIVERSIDAD DE CANTABRIA

```
   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;
```


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

¿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;
```


2.6.2. Tipos Reales

Declaración:

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

Subtipos reales:

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

2.6.3. Tipos enumerados

UC 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
```


© Javier Gutiérrez, Michael González
26/sept/08

Entrada/salida de tipos escalares

UC 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.

© Javier Gutiérrez, Michael González
26/sept/08

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;
```


© Javier Gutiérrez, Michael González
26/sept/08

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");
```


© Javier Gutiérrez, Michael González
26/sept/08

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;
```


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

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;
```


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:

UNIVERSIDAD DE CANTABRIA

- 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 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 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`

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

Arrays no restringidos

UC 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 v3: arrays no restringidos

UC 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 1..N 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 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;
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 1..N 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 1..N loop
         Get(Entrada,"V("&Integer'Image(I)&")",V1(I));
      end loop;
      -- lee V2
      Wait(Entrada,"Introduce Vector V2");
      for I in 1..N loop
         Get(Entrada,"V("&Integer'Image(I)&")",V2(I));
      end loop;
      -- calcula resultado
      for I in 1..N 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 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);
```


Ejemplo: Diagrama estadístico de barras

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;
```


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 (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

© Javier Gutiérrez, Michael González
26/sept/08

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 como declaraciones

© Javier Gutiérrez, Michael González
26/sept/08

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: 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;
```


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 ejecutarla.

Ejemplo:

UNIVERSIDAD DE CANTABRIA

```
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;
```


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;
```

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;
```


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);
   C:=Sin(B);
   A:=Sin(B,360.0);
   B:=A**C;
end Prueba;
```

2.7.3. 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 aparecen hasta el final del bloque**
- ***Regla 2*: Son visibles dentro de 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

UNIVERSIDAD DE CANTABRIA

Objeto	Visibilidad
P1	1-22
A	2-22
B	2-12, 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	18-22

```
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.**