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

6.1. Introducción

UNIVERSIDAD DE CANTABRIA

La abstracción de tipos es un requisito esencial para escribir software reutilizable

Se puede implementar en Ada mediante módulos *genéricos*:

- **El *módulo genérico* es una plantilla en la que unos parámetros genéricos quedan indeterminados**
- **Los *parámetros genéricos* pueden ser tipos de datos, valores, subprogramas, y paquetes**
- **Un módulo genérico puede ser un procedimiento, función, o paquete**
- **Para usar un módulo genérico éste debe *instanciarse*, indicando qué parámetros concretos se usarán**

Notas:

UNIVERSIDAD DE CANTABRIA

Observar la diferencia entre módulo genérico y parámetro genérico

- El módulo genérico es una parte de un programa que se define con algunos de sus elementos indeterminados; se determinarán más adelante
- Los parámetros genéricos son los elementos que han quedado indeterminados

Ejemplo

```
package Conjuntos is
   subtype Elemento is String(1..20);
   type Conjunto is private;

   function Vacio return Conjunto;
   procedure Inserta (E : Elemento;
                      C : in out Conjunto);
   procedure Extrae  (E : Elemento;
                      C : in out Conjunto);
   -- pertenencia
   function "<" (E: Elemento; C: Conjunto) return Boolean;
   function "+" (X,Y : Conjunto) return Conjunto; -- Unión
   function "*" (X,Y : Conjunto) return Conjunto; -- Intersección
   function "-" (X,Y : Conjunto) return Conjunto; -- Diferencia
   function "<" (X,Y : Conjunto) return Boolean; -- Inclusión
   No_Cabe : exception;
private
   Max_Elementos : constant Integer:=100;
   type Conjunto is ...;
end Conjuntos;
```

Comentarios sobre el ejemplo

UNIVERSIDAD DE CANTABRIA

- Para hacer un conjunto de otro tipo de datos, hay que reescribir el paquete, cambiando el tipo `Elemento`
- Si el tipo `Elemento` está declarado en un paquete externo y se desea hacer conjuntos de dos o más tipos diferentes, también será necesaria la modificación del módulo

Conclusión: El paquete `Conjuntos` no es reutilizable para diferentes tipos de elementos

6.2. Paquetes Genéricos

UNIVERSIDAD DE CANTABRIA

La solución para hacer los conjuntos independientes del tipo "**Elemento**" es la utilización de paquetes genéricos:

```
generic
   type Elemento is private;
package Conjuntos is
   type Conjunto is private;

   -- todo igual que antes
end Conjuntos;
```

Para usar esta unidad es preciso instanciarla, indicando el tipo de dato que se va a usar

```
package Conjuntos_Reales  is new Conjuntos (Float);
package Conjuntos_Enteros is new Conjuntos (Integer);
```

Notas:

El nuevo paquete Conjunto es genérico. En él queda indeterminado el tipo Elemento.

Por tanto, el tipo Elemento es el parámetro genérico.

Para usar el paquete hay que instanciarlo, indicando qué es el tipo Elemento

Pueden hacerse diversas instancias, con diversos tipos "Elemento"

Paquetes genéricos (cont.)

UNIVERSIDAD DE CANTABRIA

La unidad genérica `Conjuntos` es reutilizable

- **es independiente del tipo de dato almacenado**

Un módulo genérico también se puede hacer independiente de un valor.

- **Por ejemplo, el módulo anterior tenía una limitación de 100 elementos**
- **Esta limitación se puede eliminar de este modo:**

```
generic
   Max_Elementos : Integer;  -- nuevo parámetro genérico
   type Elemento is private;
package Conjuntos is
   -- igual que antes, sin la constante Max_Elementos
end Conjuntos;
```

Paquetes genéricos (cont.)

UNIVERSIDAD DE CANTABRIA

La declaración de esta unidad para 500 números reales y para 300 números enteros:

```
package Conjuntos_Reales  is new Conjuntos (500,Float);
package Conjuntos_Enteros is new Conjuntos (300,Integer);
```

6.3. Subprogramas genéricos

UNIVERSIDAD DE CANTABRIA

Los subprogramas genéricos necesitan una especificación separada del cuerpo

Especificación (fichero `intercambia.ads`)

```
generic
   type Dato is private;
procedure Intercambia (A,B : in out Dato);
```

Cuerpo (fichero `intercambia.adb`)

```
procedure Intercambia (A,B : in out Dato) is
   Temp : Dato:=A;
begin
   A:=B;
   B:=Temp;
end Intercambia;
```

Notas:

A todos los subprogramas (procedimientos y funciones) se les puede poner una interfaz separada del cuerpo

```
-- especificación
procedure P(I : Integer);

-- cuerpo
procedure P(I:Integer) is
  --declaraciones
begin
  -- instrucciones
end P;
```

Sin embargo, como no es obligatorio poner la interfaz, no se suele escribir

Ahora bien, para los procedimientos y funciones genéricos es obligatorio poner la interfaz separada del cuerpo

- los parámetros genéricos se especifican en la interfaz

El ejemplo de arriba muestra un procedimiento que sirve para intercambiar el valor de dos variables de un tipo indeterminado llamado Dato

Subprogramas genéricos (cont.)

UNIVERSIDAD DE CANTABRIA

Instanciación y uso:

```
with Intercambia;
procedure Prueba is
   procedure Intercambia_Enteros is new Intercambia(Integer);

   A : Integer:=...;
   B : Integer:=...;
begin
   ...
   Intercambia_Enteros(A,B);
end Prueba;
```

6.4. Tipos como Parámetros Genéricos

UNIVERSIDAD DE CANTABRIA

Es posible definir diversas categorías de tipos al definir un parámetro genérico. Algunas posibilidades:

- Tipos discretos (enteros, enumerados, o caracteres):
  - `type T is (<>);`
- Tipos enteros:
  - `type T is range <>;`
- Tipos reales:
  - `type T is digits <>;`
- Tipos array:
  - `type T is array (Indice) of Elemento;`

Tipos como Parámetros Genéricos (cont.)

- Cualquier tipo:
  - `type T is private;`
- Cualquier tipo extensible (ver tema 8):
  - `type T is tagged private;`
- Cualquier tipo derivado de otro extensible (ver tema 8):
  - `type T is new Otro_Tipo with private;`

Notas:

UNIVERSIDAD DE CANTABRIA

Según la categoría podremos usar diversas propiedades del tipo

- Para los tipos discretos podremos usar rangos, usarlos como índices de arrays o para hacer bucles de tipo for: también podemos usar operadores relacionales y asignaciones
- Para los tipos enteros podremos usar además de lo indicado para los discretos operaciones aritméticas como sumar o multiplicar
- Similarmente, para los tipos reales podremos usar además de lo indicado para los discretos operaciones aritméticas como sumar o multiplicar
- Para los tipos array podremos acceder a sus elementos y usar atributos con ’Length, ’Range, ’First, o ’Last
- Para los tipos privados sólo podemos usar asignación y comparación de igualdad o desigualdad
- Con un tipo extensible podremos extenderlo
- Para un tipo derivado de otro extensible podremos usar todas las operaciones primitivas de su antecesor y a su vez extenderlo

Los tipos extensibles (también llamados etiquetados) se verán en el tema 8.

Ejemplo: conjuntos genéricos de elementos discretos

UNIVERSIDAD DE CANTABRIA

```
generic
    type Elemento is (<>);
package Conjuntos_Discretos is
   type Conjunto is private;

   function Vacio return Conjunto;
   procedure Inserta (E : Elemento; C : in out Conjunto);
   procedure Extrae  (E : Elemento; C : in out Conjunto);
   -- pertenencia
   function "<" (E: Elemento; C: Conjunto) return Boolean;
   function "+" (X,Y : Conjunto) return Conjunto; -- Unión
   function "*" (X,Y : Conjunto) return Conjunto; -- Intersección
   function "-" (X,Y : Conjunto) return Conjunto; -- Diferencia
   function "<" (X,Y : Conjunto) return Boolean; -- Inclusión

   No_Cabe : exception;
private
    type Conjunto is array (Elemento) of Boolean;
end Conjuntos_Discretos;
```

Ejemplo (cont.)

UNIVERSIDAD DE CANTABRIA

Ejemplo de instanciación:

```
type Color is (Rojo, Verde, Amarillo, Azul);
package Colores is new Conjuntos_Discretos(Color);
use Colores;
```

Comentarios

- Si el tipo `Elemento` no fuese discreto no se podría usar como índice de un array

6.5. Subprogramas como Parámetros Genéricos

UNIVERSIDAD DE CANTABRIA

Se puede incluir un subprograma como parámetro genérico formal:

```
with procedure P (lista parámetros);
with function F (lista parámetros) return tipo;
```

Al instanciar la unidad genérica se debe incluir un parámetro actual que sea un procedimiento con el mismo perfil (mismos parámetros)

Ejemplo

UNIVERSIDAD DE CANTABRIA

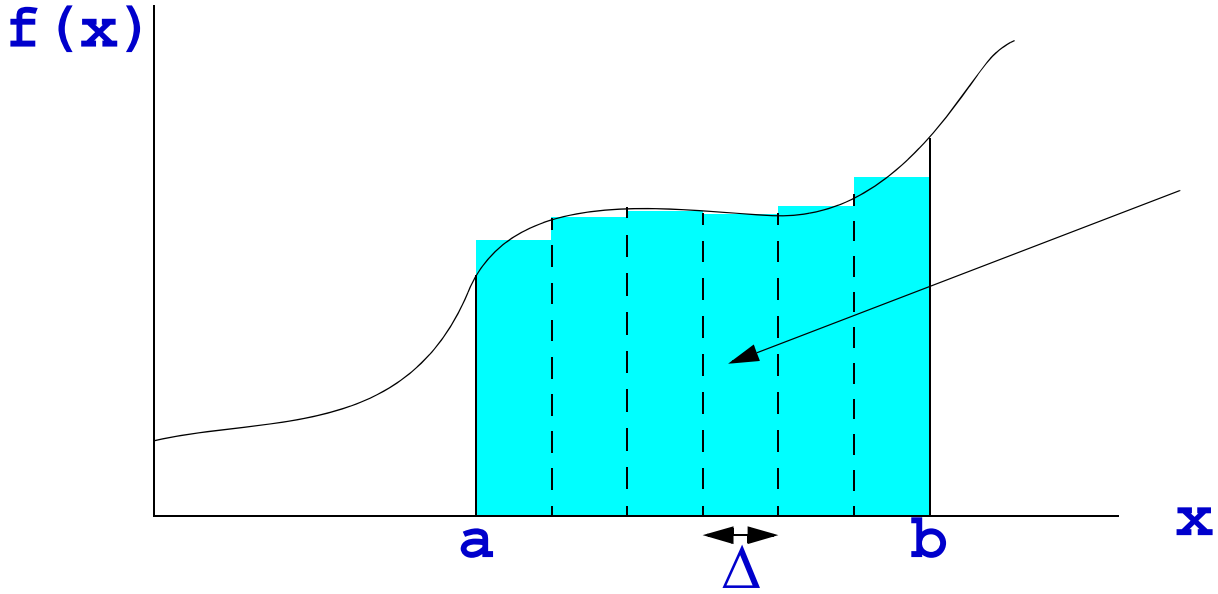
Ejemplo de una función que permite calcular la integral definida de cualquier función real entre a y b

Especificación

```
generic
    Num_intervalos : Positive;
    with function F(X : Float) return Float;
function Integral (A,B : Float) return Float;
```

Se utilizará para ello el método de integración numérica por suma de áreas de pequeños rectángulos:

Integración numérica



Area rectángulo = Δ f(x_{med}),
x_{med} : punto medio intervalo
Δ : anchura intervalo
n : Número de intervalos

Ejemplo (cont.)

UNIVERSIDAD DE CANTABRIA

Cuerpo

```
function Integral (A,B : Float) return Float is
   Delta_X   : Float :=(B-A)/Float(Num_Intervalos);
   X         : Float :=A+Delta_X/2.0;
   Resultado : Float :=0.0;
begin
   for i in 1..Num_Intervalos loop
       Resultado:=Resultado+F(X)*Delta_X;
       X := X+Delta_X;
   end loop;
   return Resultado;
end Integral;
```

Ejemplo de Uso de Función Genérica

UNIVERSIDAD DE CANTABRIA

Ejemplo del cálculo de la integral de x^2 entre 0.5 y 1.5:

```
with Integral, Ada.Text_IO, Ada.Float_Text_IO;
use Ada.Text_IO, Ada.Float_Text_IO;
procedure Prueba_Integral is

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

     function Integral_Cuad is new Integral(1000,Cuadrado);

begin
     Put("Integral de x**2 entre 0.5 y 1.5: ");
     Put(Integral_Cuad(0.5,1.5));
     New_Line;
end Prueba_Integral;
```

Otro ejemplo de instanciación

UNIVERSIDAD DE CANTABRIA

Ejemplo del cálculo de la integral de log(x) entre 2.1 y 3.2:

```
with Integral, Ada.Text_IO, Ada.Float_Text_IO,
     Ada.Numerics.Elementary_Functions;
use Ada.Text_IO, Ada.Float_Text_IO,
     Ada.Numerics.Elementary_Functions;
procedure Prueba_Integral is

     function Integral_Log  is new Integral(1000,Log);

begin
     Put("Integral de log(x) entre 2.1 y 3.2: ");
     Put(Integral_Log(2.1,3.2));
     Text_IO.New_Line;
end Prueba_Integral;
```

6.6. Punteros a subprogramas

UNIVERSIDAD DE CANTABRIA

Habitualmente se prefiere para un caso como el de la función `Integral` el uso de punteros a subprogramas

La definición de un tipo puntero a subprograma es:

```
type nombre is access function (parámetros) return tipo;

type nombre is access procedure (parámetros);
```

Permite reemplazar en algunos casos a los subprogramas usados como parámetros genéricos

Ejemplo: Integral con punteros

UNIVERSIDAD DE CANTABRIA

```
type P_Func is access function (X : Float) return Float;

function Integral
   (F : P_Func; A,B : Float;
    Num_Intervalos : Positive:=1000)
    return Float
is
   Delta_X : float := (B-A)/Float(Num_Intervalos);
   X : Float := A+Delta_X/2.0;
   Resultado : Float := 0.0;
begin
   for i in 1..Num_Intervalos loop
      Resultado:=Resultado+F(X)*Delta_X;
      X:=X+delta_X;
   end loop;
   return Resultado;
end Integral;
```

Notas:

Observar que en el ejemplo de arriba la llamada F(x) es a una función que ahora mismo no conocemos. Se determinará al invocar a Integral, mediante un puntero a una función.

- Aunque no conocemos F(X) sí sabemos, por su tipo, que requiere un parámetro real y retorna un número real

Ejemplo: Integral con punteros (cont.)

UNIVERSIDAD DE CANTABRIA

Ejemplo de cálculo de la integral de x^2 entre 0.5 y 1.5:

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

procedure Prueba_Integral is
begin
   Put("La Integral de x**2 entre 0.5 y 1.15:");
   Put(Integral(Cuadrado'Access,0.5,1.5));
   New_Line;
end Prueba_Integral;
```

Sin embargo, en otros casos el uso de subprogramas como parámetros genéricos es adecuado

- p.e., si la unidad ya tiene otros parámetros genéricos

Notas:

Observar que el puntero a la función se obtiene con el atributo Access:
Cuadrado'Access