Seminario de Programación en Ada

UNIVERSIDAD DE CANTABRIA

Anexo

- **Unidades genéricas**

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.**

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.

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);
```

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 del siguiente modo:**

```
generic
   Max_Elementos : Integer;
   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);
```

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;
```

Tipos como Parámetros Genéricos

UNIVERSIDAD DE CANTABRIA

Tipos discretos (enteros, enumerados, o caracteres):

- **type T is (<>);**

Tipos enteros:

- **type T is range <>;**

Tipos reales:

- **type T is digits <>;**

Cualquier tipo:

- **type T is private;**

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

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: Integral de función real

UNIVERSIDAD DE CANTABRIA

Especificación

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

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;
```