Seminario de Programación en Ada

Bloque II

- Paquetes
- Estructuras de datos dinámicas
- Tratamiento de errores

© Michael González, J. Javier Gutiérrez, Héctor Pérez

Paquetes Ada

UNIVERSIDAD DE CANTABRIA

Motivación:

- Encapsular juntas declaraciones de constantes, tipos y subprogramas
- *Programación orientada a objetos*: encapsulamiento de clases con especificación de una interfaz visible, y ocultando los detalles internos

Los paquetes Ada tienen dos partes:

- parte visible o especificación
  - a su vez, tiene una parte no visible o privada
- cuerpo, con los detalles internos

Declaración de paquetes

UNIVERSIDAD DE CANTABRIA

Se ponen en la parte de declaraciones de un programa o en un fichero aparte (como veremos más adelante)

Especificación

```
package Nombre is

   declaraciones de datos y tipos;
   declaraciones de tipos y constantes privadas;

   declaraciones de cabeceras de subprogramas;
   declaraciones de especificaciones de paquetes;
private
   otras declaraciones necesarias más abajo;
   declaraciones completas de los tipos y constantes privados
end Nombre;
```

Declaración de paquetes (cont.)

UNIVERSIDAD DE CANTABRIA

Cuerpo

```
package body Nombre is

   declaraciones de datos y subprogramas;
   declaraciones de subprogramas y paquetes de la
     especificación;
begin
   instrucciones de inicialización;
end Nombre;
```

Las instrucciones de inicialización

- se ejecutan una sóla vez antes de que se use el paquete
- son opcionales (junto a su `begin`)
- sirven para inicializar variables declaradas en el paquete

Uso de un paquete

UNIVERSIDAD DE CANTABRIA

Uso de objetos de la parte visible de un paquete:

```
Paquete.objeto
```

Con cláusula `use` se puede usar el objeto directamente

```
use Paquete;
```

Con una cláusula `use type` se pueden usar los operadores de un tipo directamente

```
use type Paquete.Tipo;
```

- Si no se hace, los operadores se tendrían que usar como funciones:

```
c:=Paquete."+"(a,b); -- en lugar de c:=a+b;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Ejemplo: Números complejos Especificación

UNIVERSIDAD DE CANTABRIA

```
package Complejos is

   type Complejo is private;

   function Haz_Complejo (Re,Im : Float) return Complejo;

   function Real(C : Complejo) return Float;
   function Imag(C : Complejo) return Float;

   function "+" (C1,C2 : Complejo) return Complejo;

   function Image(C : Complejo) return String;

private

   type Complejo is record
      Re,Im : Float;
   end record;

end Complejos;
```

Ejemplo: Números complejos Cuerpo

UNIVERSIDAD DE CANTABRIA

```
package body Complejos is

   function "+" (C1,C2 : Complejo) return Complejo is
   begin
      return (C1.Re+C2.Re,C1.Im+C2.Im);
   end "+";

   function Haz_Complejo (Re,Im : Float) return Complejo is
   begin
      return  (Re,Im);
   end Haz_Complejo;

   function Imag (C : Complejo) return Float is
   begin
      return C.Im;
   end Imag;
```

Ejemplo: Números complejos Cuerpo (cont'd)

UNIVERSIDAD DE CANTABRIA

```
   function Image (C : Complejo) return String is
   begin
      if C.Im>=0.0 then
         return Float'Image(C.Re)&" + "&Float'Image(C.Im)&" J";
      else
        return Float'Image(C.Re)&" - "&Float'Image(abs C.Im)&" J";
      end if;
   end Image;

   function Real (C : Complejo) return Float is
   begin
      return C.Re;
   end Real;

end Complejos;
```

Ejemplo: Números complejos Programa Principal

UNIVERSIDAD DE CANTABRIA

```
with Complejos, Ada.Text_IO;
use Ada.Text_IO;
use type Complejos.Complejo;

procedure Prueba_Complejos is
   C1,C2,C3 : Complejos.Complejo;
begin
   C1:= Complejos.Haz_Complejo(3.0,4.0);
   C2:= Complejos.Haz_Complejo(5.0,-6.0);
   Put_Line ("C1="&Complejos.Image(C1));
   Put_Line ("C2="&Complejos.Image(C2));
   C3:= C1+C2;
   Put_Line ("Parte real C3="&Float'Image(Complejos.Real(C3)));
   Put_Line ("Parte imag C3="&Float'Image(Complejos.Imag(C3)));
end Prueba_Complejos;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Compilación separada

UNIVERSIDAD DE CANTABRIA

En Ada se pueden compilar separadamente:

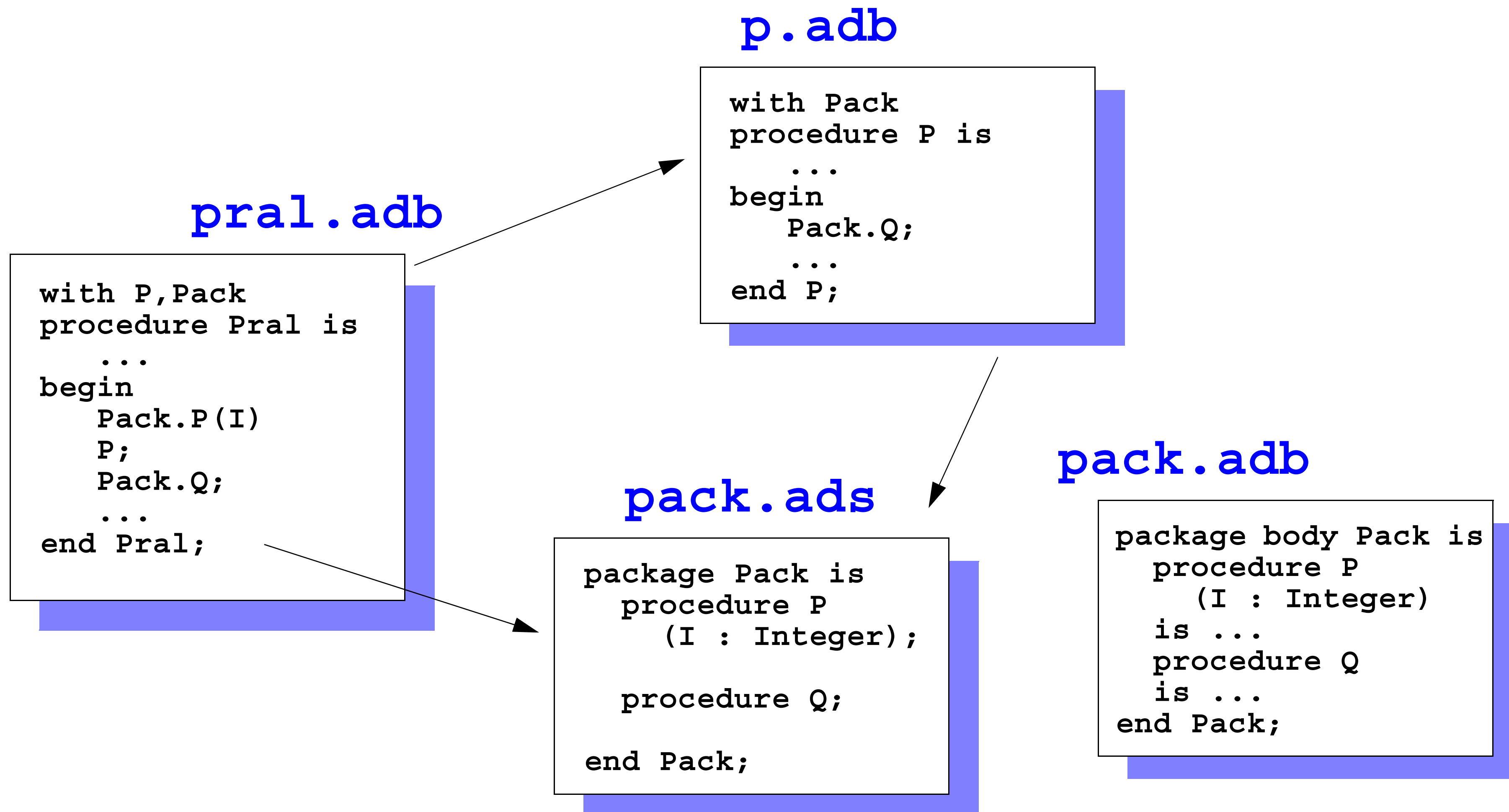
- subprogramas
- especificaciones de paquetes
- cuerpos de paquetes

Cada uno de ellos se llama "unidad de librería"

Para usar una unidad compilada separadamente se debe usar una cláusula `with`.

El orden de compilación, en algunos compiladores, es importante. En *Gnat* no importa

Ejemplo



© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Reglas de visibilidad

UNIVERSIDAD DE CANTABRIA

Quedan modificadas en los siguientes aspectos:

- Las declaraciones de la especificación (visibles y privadas) son visibles en el cuerpo
- Las declaraciones de la parte visible de la especificación son visibles en el bloque en que se declara el paquete:
  - si se antepone el nombre del paquete
  - o mediante cláusula use
- Las declaraciones del cuerpo sólo son visibles en ese cuerpo.

Paquetes hijos

UNIVERSIDAD DE CANTABRIA

Es posible crear paquetes que son "*hijos*" de otros, formando una estructura jerárquica.

Lo que caracteriza a un paquete hijo es que tiene visibilidad sobre la parte privada del padre, pero no sobre el cuerpo

Para declararlo se usa la notación "`.`" en el nombre

```
package Padre is
   ...
end Padre;

package Padre.Detalle is
   ...-- puede usar la parte privada de Padre
end Padre.Detalle;
```

En *Gnat* el fichero debe tener el nombre `padre-detalle.ads`

Tipos de paquetes

UNIVERSIDAD DE CANTABRIA

1. Paquetes sin cuerpo
2. Paquetes con objetos
   - encapsulan un objeto (un dato oculto en el cuerpo) y operaciones para manipularlo
3. Paquetes con clases
   - encapsulan un tipo de datos y operaciones para manipular objetos de ese tipo
   - el tipo de datos es privado
   - el usuario puede crear todos los objetos de ese tipo de datos que desee

Paquete sin cuerpo

UNIVERSIDAD DE CANTABRIA

```
package Constantes_Atomicas is

    Carga_Electron : constant := 1.602E-19;    -- Coulombios
    Masa_Electron  : constant := 0.9108E-30;   -- Kg
    Masa_Neutron   : constant := 1674.7E-30;   -- Kg
    Masa_Proton    : constant := 1672.4E-30;   -- Kg

    -- Este package no tiene "body"

end Constantes_Atomicas;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez

Paquete con clase: Números aleatorios

UNIVERSIDAD DE CANTABRIA

```
package Ada.Numerics.Float_Random is

   --  Basic facilities

   type Generator is limited private;

   subtype Uniformly_Distributed is
        Float range 0.0 .. 1.0;

   function Random (Gen : Generator)
        return Uniformly_Distributed;

   procedure Reset  (Gen         : Generator);
   procedure Reset  (Gen         : Generator;
                     Initiator : Integer);

   --  Advanced facilities
   ...
end Ada.Numerics.Float_Random;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Ejemplo de uso de números aleatorios

UNIVERSIDAD DE CANTABRIA

```
with Ada.Numerics.Float_Random, Text_Io, Ada.Integer_Text_Io;
use Ada.Numerics.Float_Random, Text_Io, Ada.Integer_Text_Io;
procedure Dado is

   Gen : Generator;
   Pulsada : Boolean;
   C : Character;

begin
   Put_Line("Pulsa una tecla para parar");
   Reset(Gen);
   loop
      Put(Integer(6.0*Random(Gen)+0.5));
      New_Line;
      delay 0.5;
      Get_Immediate(C,Pulsada);
      exit when Pulsada;
   end loop;
end Dado;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez

Bloque II

- Paquetes
- Estructuras de datos dinámicas
- Tratamiento de errores

Relaciones entre datos

UNIVERSIDAD DE CANTABRIA

En muchas estructuras de datos es preciso establecer *relaciones* o *referencias* entre diferentes datos.

- ahorran espacio al no repetir datos
- evitan inconsistencias

Si los datos están en un array, se pueden utilizar *cursores*

- el cursor es un entero que indica el número de la casilla del array donde está el dato

Si los datos no están en un array deben utilizarse *punteros* (si los soporta el lenguaje de programación)

- son datos especiales que sirven para apuntar o referirse a otros datos

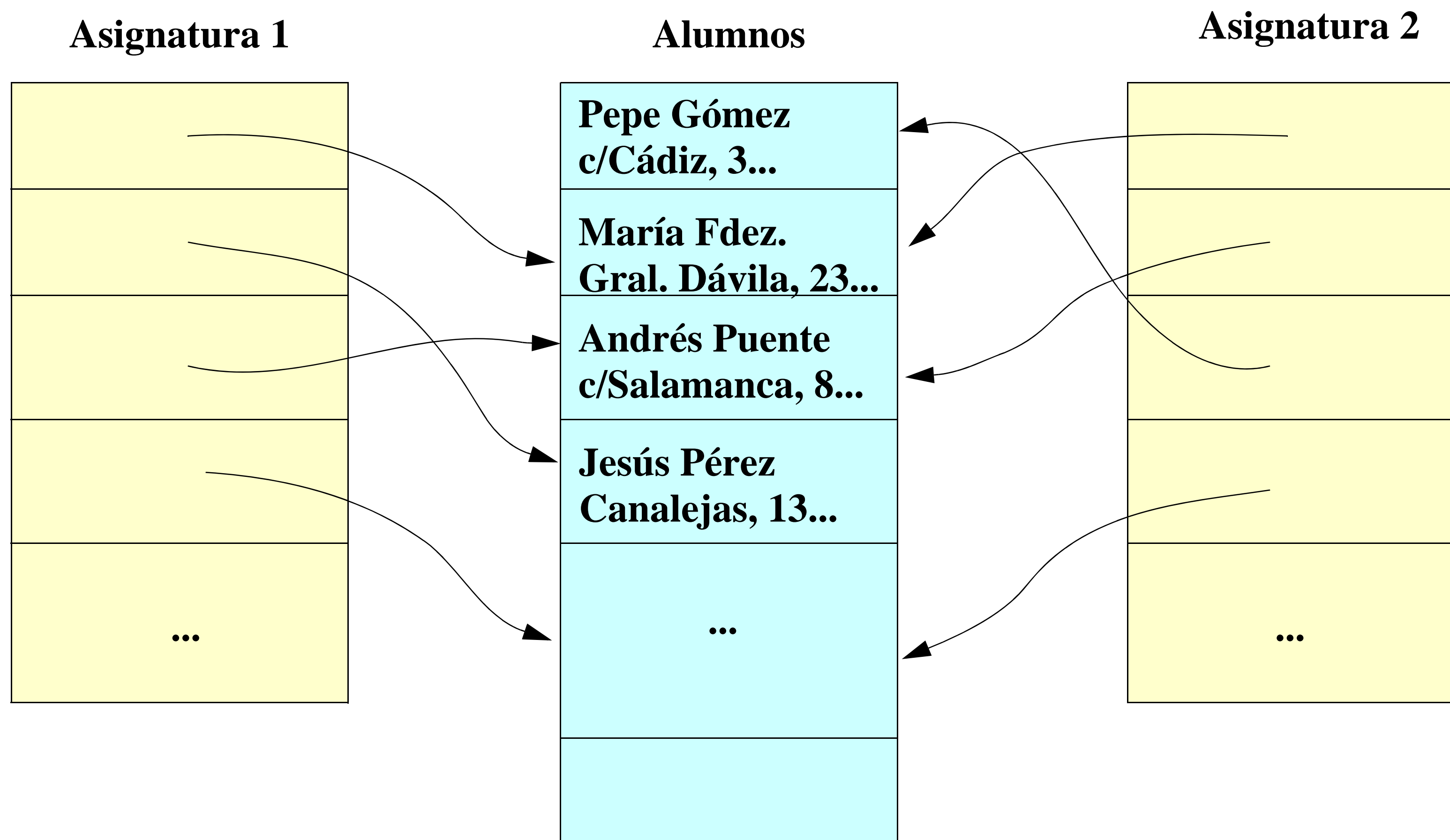
Ejemplo: listas de alumnos de asignaturas

UNIVERSIDAD DE CANTABRIA

Asignatura 1	Asignatura 2
María Fdez. Gral. Dávila, 23...	María Fdez. Gral. Dávila, 23...
Jesús Pérez Canalejas, 13...	Andrés Puente c/Salamanca, 8...
Andrés Puente c/Salamanca, 8...	Pepe Gómez c/Cádiz, 3...

¡Hay datos repetidos!

Alternativa: Referencias entre datos



© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Punteros

UNIVERSIDAD DE CANTABRIA

Los punteros o tipos acceso proporcionan acceso a otros objetos de datos

Hasta ahora el acceso era sólo por el nombre o un cursor

- ahora el puntero es más flexible

Los objetos apuntados por un puntero se pueden crear o destruir de forma independiente de la estructura de bloques

Declaración:

```
type nombre is access tipo;
```

Ejemplo:

```
type P_Integer is access Integer;
P1, P2 : P_Integer;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Punteros (cont.)

UNIVERSIDAD DE CANTABRIA

Hay un valor predefinido, `null`, que es el valor por omisión, y no se refiere a ningún dato

Creación dinámica de objetos:

```
P1:= new Integer;
P2:= new Integer'(37);
```

Uso del puntero: por el nombre

```
P1:=P2; -- copia sólo la referencia
```

Uso del valor al que apunta el puntero:

- completo: nombre_puntero.all
- campo de un registro: nombre_puntero.campo
- casilla de un array: nombre_puntero(índice)

Ejemplo

UNIVERSIDAD DE CANTABRIA

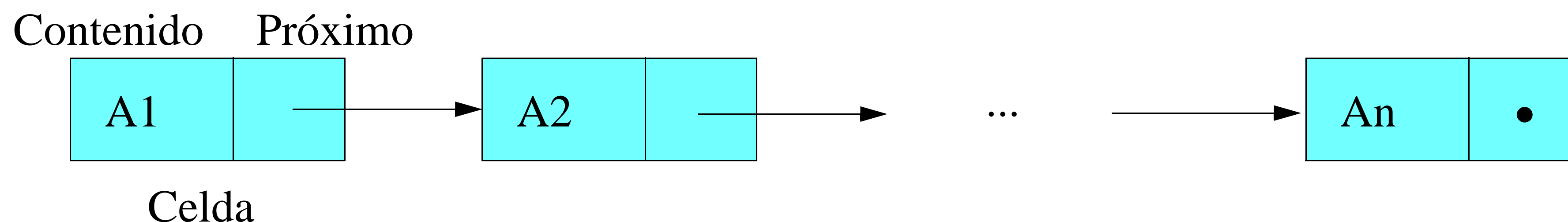
```
type Coord is record
   X,Y : Float;
end record;

type Vector is array(1..3) of Float;

type P_Coord is access Coord;
type P_Vector is access Vector;

PV1, PV2 : P_Vector;
PC1, PC2 : P_Coord;

...
PV1:=new Vector;
PV2:=new Vector'(1.0,2.0,3.0);
PV1.all:=PV2.all;    -- copia el valor completo
PV1(3):=3.0*PV2(2);  -- usa casillas individuales del array

PC1:=new Coord'(2.0,3.0);
PC1.Y:=9.0;          -- usa un campo individual del registro
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez

Estructuras de datos dinámicas

Son útiles cuando se usan para crear estructuras de datos con relaciones entre objetos.

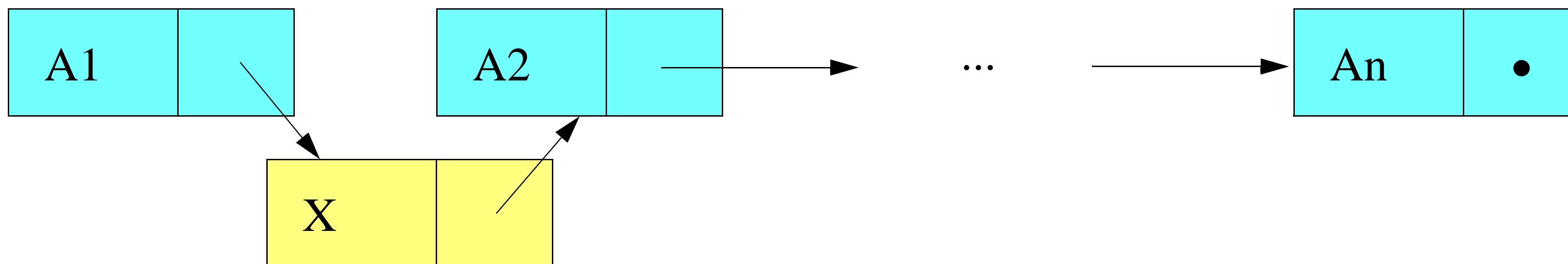
Por ejemplo, una lista enlazada



- cada elemento tiene un contenido y un puntero al próximo
- el último tiene un puntero "próximo" nulo

Flexibilidad de la estructura de datos dinámica

Se pueden insertar nuevos elementos en la posición deseada, eficientemente:



Diferencias con el array:

- los arrays tienen tamaño fijo: ocupan lo mismo, incluso medio vacíos
- con estructuras de datos dinámicas se gasta sólo lo preciso
- pero se necesita espacio para guardar los punteros

Definición de estructuras de datos dinámicas en Ada

UNIVERSIDAD DE CANTABRIA

Hay una dependencia circular entre el tipo puntero y el tipo celda

- se resuelve con una *declaración incompleta de tipo*

Ejemplo

```
type Celda; -- declaración incompleta de tipo
type P_Celda is access Celda;
type Celda is record
   Contenido : ...; -- poner el tipo deseado
   Proximo : P_Celda;
end record;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez

Ejemplo de creación de una lista enlazada

UNIVERSIDAD DE CANTABRIA

```
Lista : P_Celda;

Lista:=new Celda'(1,null); -- suponemos el contenido entero
Lista.proximo:=new Celda'(2,null);
Lista.proximo.proximo:=new Celda'(3,null);
```

Para evitar extender la notación punto indefinidamente, podemos utilizar un puntero auxiliar:

```
P : P_Celda;

P:=Lista.proximo.proximo;
P.proximo:=new Celda'(4,null);
P:=P.proximo; -- para seguir utilizando P con la siguiente celda
```

Punteros a objetos estáticos

UNIVERSIDAD DE CANTABRIA

Hasta ahora, todos los punteros lo eran a objetos creados dinámicamente (con `new`)

Es posible crear punteros a objetos (variables, constantes o subprogramas) estáticos. Ejemplos:

```
type Acceso_Entero is access all Integer;
type Acceso_Cte is access constant Integer;
```

Para poder crear un puntero a una variable estática, ésta se debe haber creado con la cláusula "`aliased`":

```
Num : aliased Integer:=5;
```

Para crear el puntero se utiliza el atributo `'Access`:

```
P : Acceso_Entero := Num'Access;
```

La principal utilidad es para llamar a funciones C

© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Punteros a objetos estáticos (cont.)

UNIVERSIDAD DE CANTABRIA

Este atributo sólo se puede usar si el tipo puntero está declarado en el mismo nivel de accesibilidad que el objeto, o en uno más profundo

- motivo: impedir tener un puntero a un objeto que ya no existe
- lo mejor es tener el objeto y el tipo puntero declarados en el mismo sitio

En caso de que sea imprescindible romper esta norma, se puede usar el atributo `'Unchecked_Access`, que también proporciona el puntero pero sin restricciones

- el uso de este atributo es peligroso

Punteros a subprogramas

UNIVERSIDAD DE CANTABRIA

Uso de subprogramas como parámetros de otros subprogramas

Utilización semejante a la de los punteros a objetos:

- Declaración del tipo puntero en el mismo nivel de accesibilidad que el subprograma
- En caso de que haya que romper esta norma, usar un puntero a subprograma anónimo

La definición de un tipo puntero a subprograma es:

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

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

Ejemplo

UNIVERSIDAD DE CANTABRIA

Ejemplo de una función que permite calcular la integral de cualquier función real

Especificación

$$\int_{inf}^{sup} f(x)dx$$

© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Ejemplo: Integral de x^2 entre 0.5 y 1.5

UNIVERSIDAD DE CANTABRIA

```
procedure Test_Integral is

   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;
```

Ejemplo: Integral de x^2 entre 0.5 y 1.5 (cont'd)

UNIVERSIDAD DE CANTABRIA

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

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

Bloque II

- Paquetes
- Estructuras de datos dinámicas
- Tratamiento de errores

Excepciones

UNIVERSIDAD DE CANTABRIA

Representan circunstancias anormales o de error

Ventajas de este mecanismo:

- El código de la parte errónea del programa está separado del de la parte correcta
- Si se te olvida tratar una excepción, te das cuenta al probar el programa, porque éste se detiene
- Se puede pasar la gestión del error de un módulo a otro de modo automático
  - agrupar la gestión del error en el lugar más apropiado

Las excepciones pueden ser predefinidas, o definidas por el usuario

Excepciones predefinidas

UNIVERSIDAD DE CANTABRIA

En el lenguaje:

- `Constraint_Error` : variable fuera de rango, índice de array fuera de rango, uso de un puntero nulo para acceder a un dato, etc.
- `Program_Error`: programa erróneo o mal construido; por ejemplo, ocurre si una función se acaba sin haber ejecutado su instrucción `return`. No se trata.
- `Storage_Error`: Memoria agotada. Ocurre tras una operación `new`, o al llamar a un procedimiento si no hay espacio en el stack.
- `Tasking_Error`: Error con las tareas concurrentes

Excepciones predefinidas (cont.)

UNIVERSIDAD DE CANTABRIA

En librerías estándares hay muchas

Un par de ellas que son frecuentes:

- `Ada.Numerics.Argument_Error`: Error con un argumento de las funciones elementales (p.e., raíz cuadrada de un número negativo)
- `Ada.Text_IO.Data_Error`: Formato de dato leído es incorrecto
  - p.e., si se leen letras pero se pretende leer un entero

Declaración de excepciones propias

UNIVERSIDAD DE CANTABRIA

Las excepciones definidas por el usuario se declaran como variables especiales, del tipo `exception`

```
nombre : exception;
```

Las excepciones se elevan, para indicar que ha habido un error:

- las predefinidas se elevan automáticamente
- las definidas por el usuario se elevan mediante la instrucción `raise`:

```
raise nombre_excepcion;
```

Una excepción que se ha elevado, se puede tratar

- Esto significa, ejecutar un conjunto de instrucciones apropiado a la gestión del error que la excepción representa

Tratamiento de excepciones

UNIVERSIDAD DE CANTABRIA

Se hace dentro de un bloque:

```
encabezamiento
   declaraciones
begin
   instrucciones
exception
   manejadores
end;
```

Los manejadores gestionan las excepciones que se producen en las instrucciones del bloque

© Michael González, J. Javier Gutiérrez, Héctor Pérez
5/mar/12

Manejadores de excepción

UNIVERSIDAD DE CANTABRIA

Los manejadores se escriben así:

```
when Nombre_Excepcion =>
   instrucciones;
```

Se pueden agrupar varias excepciones en el mismo manejador

```
when excepcion1 | excepcion2 | excepcion3 =>
   instrucciones;
```

Se pueden agrupar todas las excepciones no tratadas en un único manejador final

```
when others =>
   instrucciones;
```

- es peligroso ya que puede que el tratamiento no sea adecuado a la excepción que realmente ocurre

Funcionamiento de las excepciones

UNIVERSIDAD DE CANTABRIA

Cuando una excepción se eleva en un bloque:

a) Si hay un manejador:

- se abandonan las restantes instrucciones del bloque
- se ejecuta el manejador
- se continúa por el siguiente bloque en la secuencia de programa

b) Si no hay manejador

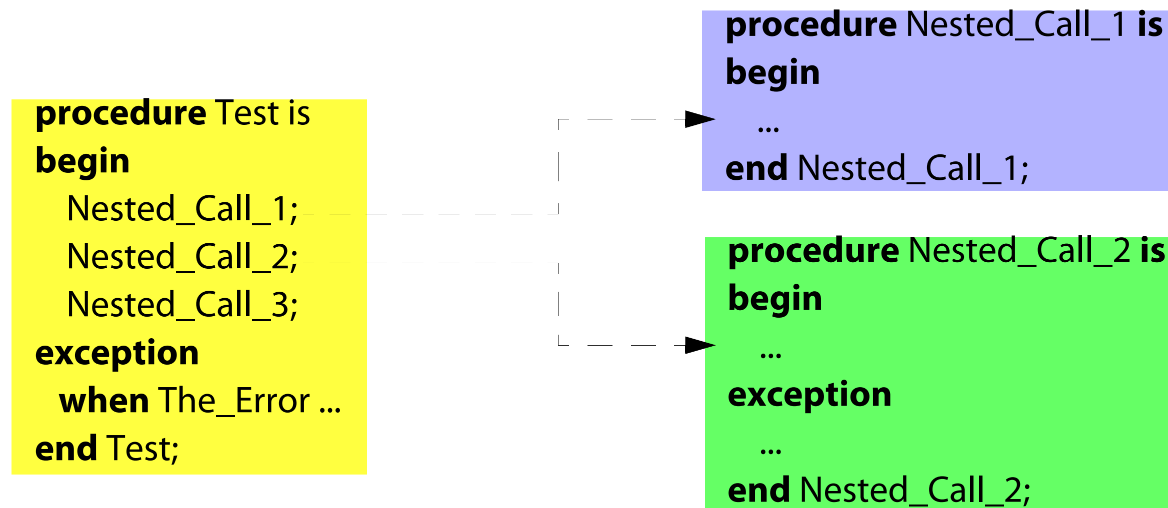
- se abandona el bloque
- se eleva la misma excepción en el siguiente bloque en la secuencia del programa

Funcionamiento de las excepciones (cont.)

UNIVERSIDAD DE CANTABRIA

El siguiente bloque en la secuencia es:

- para un subprograma, el lugar posterior a la llamada dentro del procedimiento/función invocante
- para un bloque interno (`declare-begin-end`) la instrucción siguiente al `end`
- para el programa principal, nadie
  - el programa se detiene con un mensaje de error



Creación de bloques para tratar excepciones

UNIVERSIDAD DE CANTABRIA

Si se desea continuar ejecutando las instrucciones de un bloque donde se lanza una excepción, es preciso crear un bloque más interno:

```
begin
   declare
   begin   -- del bloque interno
      instrucciones que pueden fallar
   exception
      manejadores
   end;    -- del bloque interno
   instrucciones que es preciso ejecutar, aunque haya fallo
end;
```

Paquete Ada.Exceptions (1/2)

UNIVERSIDAD DE CANTABRIA

```
package Ada.Exceptions is

   type Exception_Id is private;
   Null_Id : constant Exception_Id;

   function Exception_Name
        (X : Exception_Id) return String;

   type Exception_Occurrence is
         limited private;

   function  Exception_Message
        (X : Exception_Occurrence)
         return String;

   function Exception_Identity
        (X : Exception_Occurrence)
         return Exception_Id;
```

Paquete Ada.Exceptions (2/2)

UNIVERSIDAD DE CANTABRIA

```
   function Exception_Name
        (X : Exception_Occurrence)
         return String;

   function Exception_Information
        (X : Exception_Occurrence)
         return String;

   ...

private
   ...
end Ada.Exceptions;
```

Ejemplo: Tratamiento de excepciones (1/4)

```
--  Ada.Exceptions nos proporciona informacion sobre la excepcion
with Ada.Exceptions,Ada.Text_Io;
use Ada.Exceptions,Ada.Text_Io;

procedure Test_Exceptions is

   type Nota is range 1..10;
   --  Definimos una nueva excepcion
   Opcion_No_Definida : exception;
   Continuar : Character;
```

Ejemplo: Tratamiento de excepciones (2/4)

UNIVERSIDAD DE CANTABRIA

```
-- Subprograma que eleva excepciones
procedure Eleva_Excepcion is
   Opcion : Character;
   Mi_Nota : Nota := 5;
begin
   Put_Line ("Seleccione la opcion deseada:");
   Put_Line ("1. Elevar Constraint_Error");
   Put_Line ("2. Elevar Data_Error");
   Put_Line ("3. Elevar Storage_Error");
   New_Line;
   Put("Introduce opcion : ");
   Get(Opcion); Skip_Line;
   case Opcion is
      when '1'    => Mi_Nota := 12;
      when '2'    => raise Data_Error;
      when '3'    => raise Storage_Error;
      when others => raise Opcion_No_Definida;
   end case;
   Put_Line ("Nunca me ejecutare");
end Eleva_Excepcion;
```

Ejemplo: Tratamiento de excepciones (3/4)

```
--  Begin de Test_Exceptions
begin
   loop
      -- Nuevo bloque
      declare
      begin
         --  Invocamos metodo con excepciones definidas
         Eleva_Excepcion;
         Put_Line ("Nunca me ejecutare");
      exception
         when Error : Opcion_No_Definida =>
            null;
         when Error : Constraint_Error   =>
            Put_Line("Se ha elevado " & Exception_Name(Error));
            Put_Line("Informacion: " & Exception_Information(Error));
            Put("Desea continuar el programa? (s/n):");
            Get(Continuar); Skip_Line;
            exit when Continuar/='S' and Continuar/='s';
         when Error : others =>
            raise;
      end;
   end loop;
   Put_Line ("Programa finalizado: cuando me ejecuto?");
```

Ejemplo: Tratamiento de excepciones (4/4)

```
--  Manejador de excepciones de Test_Exceptions
exception
   when Error : Data_Error =>
      Put_Line("Se ha elevado Data_Error" );
      Put_Line("Informacion      : " & Exception_Information(Error));
   when Error : Storage_Error =>
      Put_Line("Se ha elevado Storage_Error");
      Put_Line("Informacion      : " & Exception_Information(Error));
end Test_Exceptions;
```