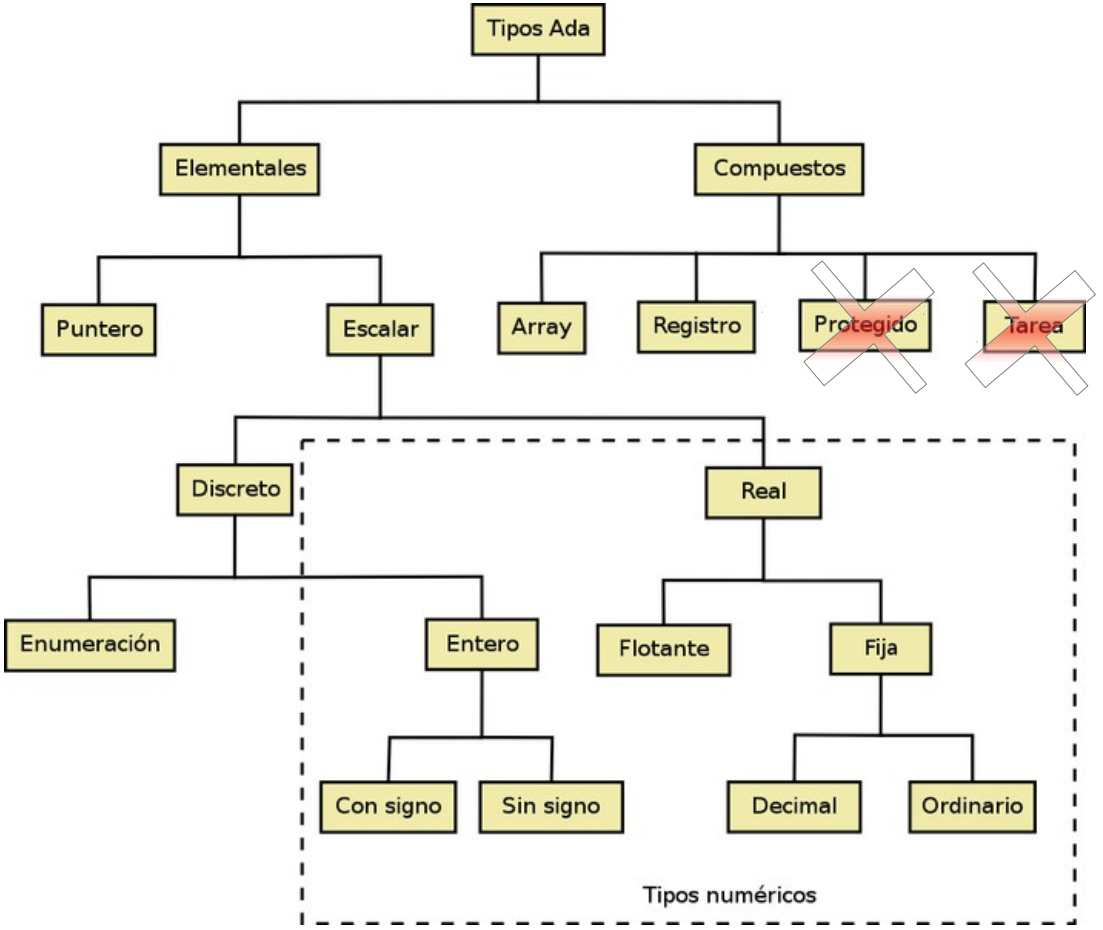
INTRODUCCIÓ AL LLENGUATGE ADA

(PACKAGES, PUNTERS I ALTRES COSES)

BIEL MOYÀ

MATERIAL EXTRET DE GUÍA MÍNIMA DE ADA

RECORDATORI - TIPUS



PACKAGES

Un package consisteix en la definició i implementació d'un conjunt d'estructures i operacions que fan feina amb aquestes.

Normalment es divideix en dues parts: especificació (fitxer amb extensió .ads) i implementació o body (fitxers amb extensió .adb)

Especificació (*.ads)

```
package nom_unitat is
    -- declaracions
    -- part visible, interfície
  private
    -- declaracions privades
    -- opcional
end nom_unitat;
```

Implementació (*.adb)

```
package body nom_unitat is
    --declaracions
    --desenvolupam procediments
    --desenvolupam funcions
end nom_unitat;
```

Les declaracions poden ser des de tipus fins prototipus de procediments o funcions. Un prototipus és la capçalera, però cambiant la paraula "**is**" per "**;**"

PACKAGES - PRIVATE

Si declaram un tipus en l'especificació d'un paquet, es podrà accedir als elements de la implementació del mateix ➡ **depenent de la implementació.**

Per evitar aquesta dependència, es declara el tipus com "**private**" en la part pública del paquet i la definició en la part privada. Així només es podrà utilitzar la assignació(:=), la comparació de igualtat (=) i la de desigualtat (/=).

```
package  Claus is
      type clau is private;
      clau_nula : constant clau;
      procedure obtenir(c : out clau);
      function "<"(x,y: in clau) return Boolean;
   private
      Max : constant := 2 ** 16 - 1;
      type clau is range 0 .. Max;
      clau_nula : constant clau := 0;
end Claus;
```

```
package body Claus is
  procedure obtenir(c : out clau) is
  begin
     -- Cos del procediment
  end obtenir;

  function "<"(x, y : in clau) return Boolean is
  begin
     -- cos de la funció
  end "<";
end Claus;
```

Si a més es volen deshabilitar les operacions d'assignació i comparació d'igualtat o desigualtat utilitzarem "**limited private**".

PACKAGES - GENÈRICS

Els packages genèrics a Ada aporten la genericitat dels tipus abstractes de dades (tad)

Es poden usar:

-Tipus privats: "private", "limited private". La definició del tipus privat correspon, en aquest cas, al que crida al paquet.

-Tipus escalars

-Arrays: És necessàri incloure com a paràmetres: el tipus dels elements de l'array, el tipus de l'índex de l'array i el tipus array:

-Punters: Especificar el tipus punter i el tipus apuntat.

-Subprogrames: S'utilitza la pataula **with** precedint l'encapçalament del subprograma que s'espera.

-Paquets: S'utiliza la paraula **with** precedint la instanciació del paquet que volem.

Exemples

```
generic
 type tElement is private;
 type index is (<>);
 type vector is array (index range <>) of tElement;
package P is
 ...
end P;
```

```
generic
type tElement is private;
with procedure suprema(x : in tElement);
package Q is
...
procedure asignar (x : in Item);
end Q;
```

PACKAGES – EXEMPLE SIMPLE

```
with ada.text_io; use ada.text_io;
with ada.integer_text_io; use ada.integer_text_io;

generic
    type item is private;
package dstack is
   type stack is limited private;
   procedure empty (s: out stack);
   procedure push(s: in out stack; x: in item);
   procedure pop (s: in out stack);
   function  top (s: in stack) return item;
   function  is_empty(s: in stack) return boolean;
private
   type cell; -- declaració incompleta
   type pcell is access cell;
   type cell is record
      x: item;
      next:pcell;
   end record;
   type stack is record
      top: pcell;
   end record;
end dstack;
```

Especificació pila (dstack.ads)

```
with Ada.Text_IO; use Ada.Text_IO;
with dstack;

procedure provapila is

package pilaInteger is new dstack (item => integer);
use pilaInteger;
pila : stack;

begin
   Put_Line("Programa prova");
   empty(pila);
   push(pila, 0);
   if is_empty(pila) then
      put("empty");
   end if;
   pop(pila);
    if is_empty(pila) then
       put("empty");
    end if;
end provapila;
```

Principal (provapila.adb)

PUNTERS (MEMORIA DINAMICA)

Els tipus punters representen direccions de memòria, s'inicialitzen automàticament a **null**.

```
type TPersona is record
        nom : string(1..20);
        cognoms : string(1..20);
end record;
```

```
type TP_Persona is access TPersona; --Tipus dels punters a TPersona
PPer1, PPer2 : TP_Persona; --Punters a TPersona
p : TPersona; --Variable  de tipus TPersona
```

Per crear una variable dinàmica s'utilitza la paraula reservada **new** seguida del tipus. Un punter no pot apuntar a variables d'un tipus si no s'ha creat així:

```
PPer1 := new TPersona; --Es crea una variable dinàmica de tipus Tpersona
PPer2 := new Tpersona'("Pep       ","Pepet     ");
```

Para accedir o modificar el contingut d'una variable referenciada por un punter, s'utilitza el qualificador **all**:

```
PPer1.all := PPer2.all;  --S'assigna a la variable referenciada per PPer1 el valor de la variable referenciada por PPer2
```

DECLARACIONS INCOMPLETES DE TIPUS

Una aplicació frequent dels punters és la creació d'estructures de dades dinàmiquess, ex: llistes encadenades.

```
type NodeLlista is record
    Info: integer;
    seguent: PNodeLlista;
end record;
```

On, PNodeLlista és el tipus dels punters a objecte de tipus NodeLlista. Això significa que ha d'existir una definició prèvia per PNodeLlista:

```
type PNodeLlista is access NodeLlista;
```


Entram en un bucle, ara no tenim NodeLlista

La solució consisteix en començar amb una declaració incompleta de NodeLlista:

```
type NodeLlista;
type PNodeLista is access NodeLlista; -- Ara ja sabem que existeix
type NodeLista is record
    Info: integer;
    Seguent: PNodeLlista;
end record;
```