Le langage Ada

Raphael Amiard - Adacore
amiard@adacore.com

- Dans les années 70, le DOD souffre d'une explosion du nombre des langages utilisés
- Il lance un concours international pour un langage qui répond à toutes ces exigences (1974)

AdaCore

- Plusieurs propositions
- Vainqueur: L'équipe de Jean Ichbiah, CII Honeywell Bull
- Première normalisation du langage (ANSI, ISO): 1983
- Révisions majeures en 1995, 2005, 2012.
- L'un des seuls langages normalisés à priori

AdaCore

- Généraliste
  - Efficacité
  - Simplicité
  - Implémentabilité

- Haut niveau de génie logiciel
  - Maintenabilité
  - Portabilité
  - Fiabilité

- Norme claire et non ambiguë
- Travail sur des plateformes embarquées
- Traitements parallèles
- Gestion des données bas niveau

AdaCore

- Dérivé d'une syntaxe type Pascal
- Impératif (comme Fortran, Cobol, C/C++, Java, Python…)
- Parallélisme intégré au langage (par opposition aux API type pthread)
- Modulaire (facilité de mise en place de sous-ensembles)
- Vérifications statiques et dynamiques (bornes…)

AdaCore

Ada aujourd'hui

Marché privilégié:

- Systèmes temps-réel
- Systèmes critiques (safety critical, mission critical)
- Systèmes de sécurité (MILS)

Exemples

- Arianne 6
- 787 Dreamliner (Common Core System)
- Airbus A350 XWB (Air Data Inertial Reference Unit)
- Sentinel 1 (Environmental Satellite System)
- Canadian Space Arm
- Meteor (metro line 14)

AdaCore

Exemple: Hello, World

```ada
with Ada.Text_IO; use Ada.Text_IO;

--  Display a welcome message
procedure Greet is
begin
    Put_Line ("Hello, world!");
end Greet;
```

```
$ gnatmake greet.adb
$ ./greet
```

AdaCore

Imperative language

Imperative language - for loops

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
begin
   for I in 1 .. 10 loop
      Put_Line("Hello, World!");
   end loop;
end Greet;
```

- I here denotes a constant that is only accessible in the loop.
- "1 .. 10" is a range.
- Put_Line is a procedure call. procedure is like a fn returning void in C/C++.

AdaCore

Imperative language - while loops

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   -- Variable declaration. Only legal in declarative
   -- parts.
   I : Integer := 1;
begin
   -- Condition. *Must* be of type boolean
   while I < 10 loop
      Put_Line("Hello, World!");

      -- Assignment
      I := I + 1;
   end loop;
end Greet;
```

AdaCore

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 1;
begin
   loop
      Put_Line("Hello, World!");
      exit when I = 5;
      --  Exit statement - takes a boolean condition
      I := I + 1;
   end loop;
end Greet;
```

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 1;
begin
   loop
      Put_Line("Hello, World!");
      if I = 5 then
         exit;
      end if;
      I := I + 1;
   end loop;
end Greet;
```

AdaCore

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 1;
begin
   loop
      if I = 5 then
         exit;
      else
         Put_Line("Hello, World!");
      end if;
      I := I + 1;
   end loop;
end Greet;
```

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 0;
begin
   loop
      if I = 5 then
         exit;
      elsif I = 0 then
         Put_Line ("Starting...");
      else
         Put_Line ("Hello, World!");
      end if;
      I := I + 1;
   end loop;
end Greet;
```

AdaCore

Imperative language - If/Else

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 1;
begin
   loop
      --  "or else" is the short circuit or operator
      if I = 5 or else I = 2 then
         exit;
      else
         Put_Line("Hello, World!");
      end if;
      I := I + 1;
   end loop;
end Greet;
```

AdaCore

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 1;
begin
   loop
      -- "and then" is the short circuit or operator
      if I < 5 and then I > 2 then
         exit;
      else
         Put_Line("Hello, World!");
      end if;
      I := I + 1;
   end loop;
end Greet;
```

Imperative language - Case statement

```
procedure Greet is
   I : Integer := 0;
begin
   loop
      -- Expression must be of a discrete type. All the
      -- values must be covered.
      case I is
         when 0 => Put_Line ("Starting...");
         when 3 .. 4 => Put_Line ("Hello");
         when 7 | 9 => exit;
         -- 'when others' must be the last one and alone (if
         -- present)
         when others => Put_Line ("Hello, World!");
      end case;
      I := I + 1;
   end loop;
end Greet;
```

Quizz: Imperative language

Quizz 1: Is there a compilation error?

```
for I in 10 .. 1 loop
    Put_Line("Hello, World!");
end loop;
```

Quizz 2: Is there a compilation error?

```
for I in reverse 1 .. 10 loop
    Put_Line("Hello, World!");
end loop;
```

Quizz 3: Is there a compilation error?

```ada
procedure Hello is
   I : Integer;
begin
   for I in 1 .. 10 loop
      Put_Line ("Hello, World!");
   end loop;
end Hello;
```

AdaCore

Quizz 4: Is there a compilation error?

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   I : Integer;
begin
   while I < 10 loop
      Put_Line("Hello, World!");
      I := I + 1;
   end loop;
end Greet;
```

Quizz 5: Is there a compilation error?

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
    I : Integer := 2;
begin
    while i < 10 loop
        Put_Line ("Hello, World!");
        i := i + 1;
    end loop;
end Greet;
```

Quizz 6: Is there a compilation error?

```ada
with Ada.Text_IO; use Ada.Text_IO;
with Tools;

procedure Greet is
begin
    loop
        Put_Line("Hello, World!");
        Tools.My_Proc;
    end loop;
end Greet;
```

Quizz 7: Is there a compilation error?

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 0;
begin
   loop
      if I = 5 then
         exit;
      else
         if I = 0 then
            Put_Line ("Starting...");
         else
            Put_Line ("Hello, World!");
         end if;
      end if;
      I := I + 1;
   end loop;
end Greet;
```

Quizz 8: Is there a compilation error?

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 0;
begin
   loop
      case I is
         when 0 =>
            Put_Line ("Starting...");
         when 1 .. 4 =>
            Put_Line ("Hello");
         when 5 =>
            exit;
      end case;
      I := I + 1;
   end loop;
end Greet;
```

Quizz 9: Is there a compilation error?

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
begin
   loop
      case I is
         when 0 =>
            Put_Line ("Starting...");
         when 1 .. 4 =>
            Put_Line ("Hello");
         when others =>
            exit;
      end case;
      I := I + 1;
   end loop;
end Greet;
```

AdaCore

Quizz 10: Is there a compilation error?

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   I : Integer := 0;
begin
   loop
      case I is
         when Integer'First .. 1 =>
            Put_Line ("Starting...");
         when 1 .. 4 =>
            Put_Line ("Hello");
         when others =>
            exit;
      end case;
      I := I + 1;
   end loop;
end Greet;
```

Quizz 11: Which one is an error?

```
V    : Integer;
1V   : Integer;
V_   : Integer;
_V   : Integer;
V__1 : Integer;
V_1  : Integer;
```

Strongly typed language

What is a type?

Integers

- Integer types are just regular types (not a built-in)

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type My_Int is range 1 .. 20;
   -- Declare a signed integer type, and give the bounds

   -- Like variables, declarations can only happen in
   -- declarative region
begin
   for I in My_Int loop
      Put_Line("Hello, World!");
   end loop;
end Greet;
```

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type My_Int is range 1 .. 20;
begin
   --  Iterates on every value in the range of My_Int
   for I in My_Int loop
      if I = My_Int'Last then
      --              ^ Higher bound of the type
      --  'First is the lower bound
         Put_Line ("Bye");
      else
         Put_Line("Hello, World!");
      end if;
   end loop;
end Greet;
```

```
procedure Greet is
   A : Integer := Integer'Last;
   B : Integer;
begin
   B := A + 5;
   -- This operation will overflow, eg. it will
   -- raise an exception at runtime.
end Greet;
```

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type My_Int is range 1 .. 20;
   A : My_Int := 12;
   B : My_Int := 15;
   M : My_Int := (A + B) / 2;
   -- No overflow here, overflow checks are done at
   -- specific boundaries.
begin
   for I in 1 .. M loop
      Put_Line("Hello, World!");
   end loop;
end Greet;
```

Enumerations

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);
   --  An enumeration type
begin
   for I in Days loop
      case I is
         when Saturday .. Sunday =>
            Put_Line ("Week end!");

            --  Completeness checking on enums
         when others =>
            Put_Line ("Hello on " & Days'Image (I));
            --  'Image attribute, converts a value to a
            --  String
      end case;
   end loop;
end Greet;
```

AdaCore

Strong typing

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   -- Declare two signed types
   type Meters is range 0 .. 10_000;
   type Miles is range 0 .. 5_000;

   Dist_Us : Miles;
   -- Declare a constant
   Dist_Eu : constant Meters := 100;
begin
   -- Not correct: types mismatch
   Dist_Us := Dist_Eu * 1609 / 1000;
   Put_Line (Miles'Image (Dist_Us));
end Greet;
```

Strong typing

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Conv is
   type Meters is range 0 .. 10_000;
   type Miles is range 0 .. 5_000;
   Dist_Us : Miles;
   Dist_Eu : constant Meters := 100;
begin
   Dist_Us := Miles (Dist_Eu * 1609 / 1000);
   --           ^ Type conversion, from Meters to Miles
   --  Now the code is correct

   Put_Line (Miles'Image (Dist_Us));
end;
```

Strong typing

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   C : Character;
   --  ^ Built-in character type (it's an enum)
begin
   C := '?';
   --   ^ Character literal (enumeration literal)

   C := 64;
   --   ^ Invalid: 64 is not an enumeration literal
end Greet;
```

Strong typing

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   C : Character;
begin
   C := '?';
   Put_Line ("""Ascii"" code of '" & C & "' is"
   --            ^ Use "" to insert quote in a string
              & Integer'Image (Character'Pos (C)));
   --                                   ^ 'Pos converts a
   --                                     value to its position

   C := Character'Val (64);
   --              ^ 'Val converts a position to its value
end Greet;
```

AdaCore

Subtypes

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Greet is
   type Days is (Monday, Tuesday, Wednesday, Thursday,
                 Friday, Saturday, Sunday);

   --  Declaration of a subtype
   subtype Weekend_Days is Days range Saturday .. Sunday;
   --                                 ^ Constraint of the subtype
begin
   for I in Days loop
      case I is
         --  Just like a type, a subtype can be used as a
         --  range
         when Weekend_Days =>
            Put_Line ("Week end!");
         when others =>
            Put_Line ("Hello on " & Days'Image (I));
      end case;
   end loop;
end Greet;
```

A subtype doesn't define a type

- All subtypes are of the same type.

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type Days is (Monday, Tuesday, Wednesday, Thursday,
                 Friday, Saturday, Sunday);

   subtype Weekend_Days is Days range Saturday .. Sunday;
   Day : Days := Saturday;
   Weekend : Weekend_Days;
begin
   Weekend := Day;
   --          ^ Correct: Same type
   Weekend := Monday;
   --           ^ Wrong value for the subtype
   --  Compiles, but exception at runtime
end Greet;
```

Quizz: Types

Quizz 1: Is there a compilation error?

```ada
type My_Int is range 1 .. 20.5;
```

Quizz 2: Is there a compilation error?

```ada
type My_Int is range 1 .. 20.0;
```

Quizz 3: Is there a compilation error?

```ada
A : Integer := 5;
type My_Int is range A .. 20;
```

```
type My_Int is range 1 .. Integer'Last;
```

Quizz 5: Is there a compilation error?

```ada
type My_Int_1 is range 1 .. Integer'Last;
type My_Int_2 is range Integer'First .. 0;
type My_Int_3 is range My_Int_2'First .. My_Int_2'Last;
```

Quizz 6: Is there a compilation error?

```ada
type My_Int_1 is range 1 .. Integer'Last;
subtype My_Int_2 is My_Int_1 range 1 .. 100;

V1 : My_Int_1 := 5;
V2 : My_Int_2;
V2 := V1;
```

Quizz 7: Is there a compilation error?

```ada
type My_Int_1 is range 1 .. Integer'Last;
type My_Int_2 is range 1 .. 100;

V1 : My_Int_1 := 5;
V2 : My_Int_2;
V2 := V1;
```

Quizz 8: Is there a compilation error?

```ada
type Enum is (E1, E2);
type Enum2 is (E2, E3);
```

Quizz 9: Is there a compilation error?

```ada
type Bit is ('0', '1');
```

Arrays

Array type declaration

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

procedure Greet is
   type My_Int is range 0 .. 1000;
   type Index is range 1 .. 5;

   type My_Int_Array is array (Index) of My_Int;
   --                                       ^ Type of elements
   --                           ^ Bounds of the array
   Arr : My_Int_Array := (2, 3, 5, 7, 11);
begin
   for I in Index loop
      Put (My_Int'Image (Arr (I)));
   end loop;
   New_Line;
end Greet;
```

AdaCore

Array type declaration

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

procedure Greet is
   type My_Int is range 0 .. 1000;
   type Index is range 1 .. 5;
   type My_Int_Array is array (Index) of My_Int;

   Arr : My_Int_Array := (2, 3, 5, 7, 11);
   --                     ^ Aggregate (array literal)
begin
   for I in Index loop
      Put (My_Int'Image (Arr (I)));
   end loop;
   New_Line;
end Greet;
```

AdaCore

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type My_Int is range 0 .. 1000;
   type Index is range 11 .. 15;
   --                    ^ Low bound can be any value
   type My_Int_Array is array (Index) of My_Int;
   Tab : My_Int_Array := (2, 3, 5, 7, 11);
begin
   for I in Index loop
      Put (My_Int'Image (Tab (I)));
   end loop;
   New_Line;
end Greet;
```

Array index

```ada
procedure Greet is
   type My_Int is range 1 .. 31;
   type Month is (Jan, Feb, Mar, Apr, May, Jun,
                  Jul, Aug, Sep, Oct, Nov, Dec);

   type My_Int_Array is array (Month) of My_Int;
   --                           ^ Can use an enum as the
   --                             index

   Tab : constant My_Int_Array :=
   --      ^ constant is like a variable but cannot be
   --        modified
     (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
   -- Maps months to number of days

   Feb_Days : My_Int := Tab (Feb);
   -- Number of days in February
begin
   for I in Month loop
      Put_Line (My_Int'Image (Tab (I)));
   end loop;
end Greet;
```

AdaCore

Indexation

```
procedure Greet is
   type My_Int is range 0 .. 1000;
   type Index is range 1 .. 5;
   type My_Int_Array is array (Index) of My_Int;
   Tab : My_Int_Array := (2, 3, 5, 7, 11);
begin
   Indexation
   for I in 2 .. 6 loop
      Put (My_Int'Image (Tab (I)));
      --                         ^ Indexation
      --                         ^ Will raise an exception when
      --                         I = 6
   end loop;
   New_Line;
end Greet;
```

Indexation

```ada
procedure Greet is
   type My_Int is range 0 .. 1000;
   type Index is range 1 .. 5;
   type My_Int_Array is array (Index) of My_Int;
   Tab : My_Int_Array := (2, 3, 5, 7, 11);
begin
   for I in Integer range 1 .. 5 loop
   --        ^ We say that the type of I is Integer
      Put (My_Int'Image (Tab (I)));
   --                        ^ Compile time error
   end loop;
   New_Line;
end Greet;
```

Shortcut for index

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type My_Int is range 0 .. 1000;
   type My_Int_Array is array (1 .. 5) of My_Int;
   --                                 ^ Subtype of Integer
   Tab : My_Int_Array := (2, 3, 5, 7, 11);
begin
   for I in 1 .. 5 loop
   --        ^ Likewise
      Put (My_Int'Image (Tab (I)));
   end loop;
   New_Line;
end Greet;
```

Range attribute

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type My_Int is range 0 .. 1000;
   type My_Int_Array is array (1 .. 5) of My_Int;
   Tab : My_Int_Array := (2, 3, 5, 7, 11);
begin
   for I in Tab'Range loop
   --            ^ Gets the range of Tab
      Put (My_Int'Image (Tab (I)));
   end loop;
   New_Line;
end Greet;
```

AdaCore

Unconstrained arrays

```ada
procedure Greet is
   type Days is (Monday, Tuesday, Wednesday,
      Thursday, Friday, Saturday, Sunday);

   type Workload_Type is array (Days range <>) of Natural;
   --  Indefinite array type
   --                            ^ Bounds are of type Days,
   --                              but not known

   Workload : constant Workload_Type (Monday .. Friday) :=
   --                                 ^ Specify the bounds
   --                                   when declaring
      (Friday => 7, others => 8);
   --               ^ Default value
   -- ^ Specify element by name of index
begin
   for I in Workload'Range loop
      Put_Line (Integer'Image (Workload (I)));
   end loop;
end Greet;
```

AdaCore

Declaring arrays

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);

   type Workload_Type is array (Days range <>) of Natural;

   Workload : constant Workload_Type :=
      (Monday .. Thursday => 8, Friday => 7);
   --  ^ More powerful association by name
   --  Here, no need to specify the bounds of the array
begin
   for I in Workload'Range loop
      Put_Line (Integer'Image (Workload (I)));
   end loop;
end Greet;
```

AdaCore

Predefined array type: String

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   Message : String (1 .. 11) := "dlroW olleH";
   --           ^ Pre-defined array type.
   --             Component type is Character
begin
   for I in reverse 1 .. 11 loop
            ^ Iterate in reverse order
     Put (Message (I));
   end loop;
   New_Line;
end Greet;
```

Predefined array type: String

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   Message : constant String := "Hello World";
   --                    ^ Bounds are automatically computed
   --                      from initialization value
begin
   for I in reverse Message'First .. Message'Last loop
   --                        ^ 'First and 'Last return the low and
   --                          high bound
   -- (But you should use 'Range most of the time)
      Put (Message (I));
   end loop;
   New_Line;
end Greet;
```

Declaring arrays

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);

   subtype Day_Name is String (1 .. 2);
   --  Subtype of string with known size

   type Days_Name_Type
   is array (Days) of Day_Name;
   --          ^ Type of the index
   --                    ^ Type of the element. Must be
   --                      definite
begin
   null;
end Greet;
```

Declaring arrays

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);
   subtype Day_Name is String (1 .. 2);
   type Days_Name_Type is array (Days) of Day_Name;
   Names : constant Days_Name_Type :=
     ("Mo", "Tu", "We", "Th", "Fr", "Sa", "Su");
   --  Initial value given by aggregate
begin
   for I in Names'Range loop
      Put_Line (Names (I));
   end loop;
end Greet;
```

AdaCore

Declaring arrays

```ada
-- Those two declarations produce the same thing
A : String := "Hello World";
B : String (1 .. 11) := ('H', 'e', 'l', 'l', 'o', ' ',
                         'W', 'o', 'r', 'l', 'd');
```

Quizz: Arrays

Quizz 1: Is there a compilation error ?

```ada
-- Natural is a pre-defined subtype.
subtype Natural is Integer range 0 ..  Integer'Last

type Arr is array (Natural) of Integer;
Name : Arr;
```

Quizz 2: Is there a compilation error ?

```ada
type Arr is array (Natural range <>) of Integer;
Name : Arr;
```

Quizz 3: Is there a compilation error ?

```ada
type Str_Array is array (1 .. 10) of String;
```

```
A : constant Integer := 5;
```

Quizz 5: Is there a compilation error ?

```
A : constant String (1 .. 12);
```

Quizz 6: Is there a compilation error ?

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);
   type WorkLoad_Type is array (Days range <>) of Natural;
   Workload : constant WorkLoad_Type :=
      (Monday .. Friday => 8, Friday => 7, Saturday | Sunday => 0);
begin
   for I in Workload'Range loop
      Put_Line (Integer'Image (Workload (I)));
   end loop;
end Greet;
```

Modular/Structured programming

Packages

```
package Week is
end Week;
```

Packages:

- Group related declarations together
- Define an interface (API)
- Hide the implementation
- Provide a name space

Packages

```
package Week is

   -- This is a declarative part. You can put only
   -- declarations here, no statements

   type Days is (Monday, Tuesday, Wednesday,
      Thursday, Friday, Saturday, Sunday);
   type WorkLoad_Type is array (Days range <>) of Natural;
   Workload : constant Workload_Type :=
      (Monday .. Friday => 8, Friday => 7, Saturday | Sunday => 0);
end Week;
```

Different from header files in C/C++ because:

- Language level mechanism (not a preprocessor)
- Not text based
- With'ing a package does not "copy/paste" the content of the spec into your file
- With GNAT, packages specs go in .ads files (here, it would be week.ads)

AdaCore

With-ing a package

```ada
with Ada.Text_IO; use Ada.Text_IO;

with Week;
--  references the package, and adds a dependency on
--  that package

procedure Greet is
begin
   for I in Week.Workload'Range loop
   --         ^ We reference items of the package by prefixing
   --           by package name
      Put_Line (Integer'Image (Workload (I)));
   end loop;
end Greet;
```

Using a package

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

with Week;
use Week;
-- Brings the content of the package in the current
-- namespace

procedure Greet is
begin
   for I in Workload'Range loop
   --        ^ We can reference items of the package directly
   --            now
      Put_Line (Integer'Image (Workload (I)));
   end loop;
end Greet;
```

Package body

```
package body Week is

   -- The body contains additional declarations, not
   -- visible from the spec, or anywhere outside of the
   -- body
   type WorkLoad_Type is array (Days range <>) of Natural;
   Workload : constant Workload_Type :=
      (Monday .. Friday => 8, Friday => 7, Saturday | Sunday => 0);

   function Get_Workload (Day : Days) return Natural is
   begin
      return Workload (Day);
   end;
   -- This is a function. It has a return statement
   -- (mandatory for functions)
end Week;
```

- With GNAT, packages bodies go in .adb files (here, it would be week.adb)

Subprograms

```ada
with Ada.Text_IO; use Ada.Text_IO;

-- Here we declare and define a procedure without
-- parameters
procedure Greet is
begin
   Put_Line("Hello, World!");
end Greet;
```

Subprograms

```ada
package Week is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);

   function Get_Workload (Day : Days) return Natural;
   --  We declare (but don't define) a function with one
   --  parameter, returning a Natural integer
end Week;
```

```
package Week is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);

   function Get_Day_Name
     (Day : Days := Monday) return String;
   --                              ^ We can return any type,
   --                                even indefinite ones
   --          ^ Default value for parameter
end Week;
```

```ada
package body Week is
   -- Implementation of the Get_Day_Name function
   function Get_Day_Name (Day : Days := Monday) return String is
   begin
      case Day is
         when Monday => return "Monday";
         when Tuesday => return "Tuesday";
         ...
         when Sunday => return "Sunday";
      end case;
   end Get_Day_Name;
end Week;
```

AdaCore

Parameters modes

```ada
package Week is
   type Days is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);
   procedure Next_Day (Res : out Days;
   --                       ^ Mode out: not initialized at
   --                         the beginning, procedure body can
   --                         assign it. (Like a return value)
                       Day : Days);
   --                       ^ No mode: defaults to "in",
   --                         which is like a constant
end Week;
```

Parameters modes

```
--              You can declare several params at the same
--          v time
procedure Swap (A, B : in out Integer)
--                  ^ In out is initialized at the
--                    beginning with value passed by
--                    caller. procedure can assign it,
--                    caller will get new value. Like
--                    pass by reference
--                    Use it instead of a pointer in C
is
   Tmp : Integer;
begin
   Tmp := A;
   A := B;
   B := Tmp;
   return;
   -- Return statement. Optional for procedures
end Swap;
```

Subprogram call

```ada
procedure Test_Swap
is
   X, Y : Integer;
begin
   X := 5;
   Y := 7;
   Swap (X, Y);
   --    ^ Positional parameters
   Swap (A => X, B => Y);
   --      ^ Named parameters
   Swap (B => X, A => Y);
   --      ^ You can reverse the order
end Test_Swap;
```

Function calls

```
function Double (I : Integer) return Integer is
begin
   return I * 2;
end Double;

function Quadruple (I : Integer) return Integer is
   Res : Integer := Double (Double (I));
   --                ^ Calling the double function
begin
   Double (I);
   -- ILLEGAL: You cannot ignore a function's return value

   return Res;
end Quadruple;
```

Function calls

```ada
function Pi return Float is
--          ^ No parameters, no parentheses
begin
   return 3.1419;
end Pi;

A : Float := Pi;
--          ^ No parameters, no parentheses at call site
--            either!
```

Mutually recursive subprograms

```ada
procedure Compute_A (V : Natural);
--  Forward declaration of Compute_A

procedure Compute_B (V : Natural) is
begin
   if V > 5 then
      Compute_A (V - 1);
   -- ^ Call to Compute_A
   end if;
end Compute_B;

procedure Compute_A (V : Natural) is
begin
   if V > 2 then
      Compute_B (V - 1);
   -- ^ Call to Compute_B
   end if;
end Compute_A;
```

AdaCore

Nested subprograms

```ada
function Quadruple (I : Integer) return Integer is

   function Double (I : Integer) return Integer is
   begin
      return I * 2;
   end Double;
   -- Nested function

begin
   return Double (Double (I));
end Quadruple;
```

Quizz 1: Is there a compilation error ?

```ada
package My_Type is
   type My_Type is range 1 .. 100;
end My_Type;
```

Quizz 2: Is there a compilation error ?

```ada
package Pkg is
   function F (A : Integer);
end Pkg;
```

Quizz 3: Is there a compilation error ?

```ada
package Pkg is
   function F (A : Integer) return Integer;
   function F (A : Character) return Integer;
end Pkg;
```

Quizz 4: Is there a compilation error ?

```ada
package Pkg is
   function F (A : Integer) return Integer;
   procedure F (A : Character);
end Pkg;
```

Quizz 5: Is there a compilation error ?

```
package Pkg is
   subtype Int is Integer;
   function F (A : Integer) return Integer;
   function F (A : Int) return Integer;
end Pkg;
```

AdaCore

Quizz 6: Is there a compilation error ?

```ada
package Pkg is
   procedure Proc (A : Integer);
   procedure Proc (A : in out Integer);
end Pkg;
```

Quizz 7: Is there a compilation error ?

```ada
package Pkg is
   procedure Proc (A : in out Integer := 7);
end Pkg;
```

Quizz 8: Is there a compilation error ?

```ada
package Pkg is
   procedure Proc (A : Integer := 7);
end Pkg;

package body Pkg is
   procedure Proc (A : Integer) is
      ...
   end Proc;
end Pkg;
```

Quizz 9: Is there a compilation error ?

```ada
package Pkg is
   procedure Proc (A : in out Integer);
end Pkg;

package body Pkg is
   procedure Proc (A : in out Integer) is
      ...
   end Proc;

   procedure Proc (A : in out Character) is
      ...
   end Proc;
end Pkg;
```

Quizz 10: Is there a compilation error ?

```ada
package Pkg is
   procedure Proc (A : in Integer);
end Pkg;

package body Pkg is
   procedure Proc (A : in Integer);
   procedure Proc (A : in Integer) is
      ...
   end Proc;
end Pkg;
```

Quizz 11: Is there a compilation error ?

```
-- pkg1.ads
package Pkg1 is
    procedure Proc;
end Pkg1;

-- pkg2.ads
with Pkg1;
package Pkg2 is
...
end Pkg2;

-- main.adb
with Pkg2;

procedure Main is
begin
   Pkg1.Proc
end Main;
```

AdaCore

Quizz 12: Is there a compilation error ?

```ada
-- pkg1.ads
package Pkg1 is
  procedure Proc;
  ...
end Pkg1;

-- pkg2.ads
with Pkg1; use Pkg1;
package Pkg2 is
  ...
end Pkg2;

-- pkg2.adb
package body Pkg2 is
   procedure Foo is
   begin
      Proc;
   end Foo;
end Pkg2;
```

AdaCore

Quizz 13: Is there a compilation error ?

```
-- pkg1.ads
package Pkg1 is
procedure Proc;
   ...
end Pkg1;

-- pkg2.ads
with Pkg1; use Pkg1;
package Pkg2 is
   ...
end Pkg2;

-- pkg2.adb
with Pkg1; use Pkg1;
package body Pkg2 is
   ...
end Pkg2;
```

Quizz 14: Is there a compilation error ?

```ada
-- pkg1.ads
package Pkg1 is
   procedure Proc;
   ...
end Pkg1;

-- pkg2.ads
with Pkg1;
package Pkg2 is
   ...
end Pkg2;

use Pkg1;

-- pkg2.adb
package body Pkg2 is
   procedure Foo is
   begin
      Proc;
   end Foo;
end Pkg2;
```

More about types

Array

```
Len : Natural := F (5);

--  The size of this array is not known at compile time. But the bounds are
--  fixed.
Buf : String (1 .. Len);

--  This does not change the size of the array.
Len := 3;
```

Array slices

```ada
procedure Main is
    Buf : String := "Hello ...";

    Full_Name : "Raphael Amiard";
begin
    Buf (7 .. 9) := "Bob";
    Put_Line (Buf);  --  Prints "Hello Bob"

    Put_Line ("Hi " & Full_Name (1 .. 7)); --  Prints "Hi Raphael"
end;
```

Records

```ada
type Date is record
   -- The following declarations are components of the record
   Day   : Integer range 1 .. 31;
   Month : Month_Name;
   Year  : Integer range 1 .. 3000; -- You can add custom constraints on fields
end record;
```

Records - default values

```ada
type Date is record
   Day   : Integer range 1 .. 31;
   Month : Month_Name := January;

   -- This component has a default value
   Year  : Integer range 1 .. 3000 := 2012;
   --                                  ^ Default value
end record;
```

```
Today    : Date := (31, November, 2012);
Birthday : Date := (Day => 30, Month => February, Year => 2010);
--                   ^ By name
```

```
Today.Day := 29;
```

Access types (pointers)

```ada
-- Declare an access type
type Date_Acc is access Date;

D : Date_Acc;
D := null;
--   ^ Literal for "access to nothing"
```

```ada
type Date_Acc is access Date;
D : Date_Acc;

Today : Date := D.all; -- Access dereference
J     : Day := D.Day
--             ^ Implicit dereference for record and array components
--               Equivalent to D.all.day
```

```ada
D : Date_Acc := new Date;  --  Allocation (using default values)
```

Allocation (by type)

```
type String_Acc is access String;
--                      ^ Access to unconstrained array type
Msg : String_Acc;
--    ^ Default value is null

Buffer : String_Acc := new String (1 .. 10);
--                              ^ Constraint required
```

```
D   : Date_Acc := new Date'(30, November, 2011);
Msg : String_Acc := new String'("Hello");
```

Mutually recursive types

```ada
type Node;
--  This is an incomplete type declaration, it must be
--  completed in the same declarative region.

type Node_Acc is access Node;

type Node is record
   Content    : Natural;
   Prev, Next : Node_Acc;
end record;
```

More about records

```
Max_Len : constant Natural := Compute_Max_Len;
--                            ^ Not known at compile time

type Person is record
   First_Name : String (1 .. Max_Len);
   Last_Name  : String (1 .. Max_Len);
end record;
```

Records with discriminant

```
type Person (Max_Len : Natural) is record
--              ^ Discriminant. Cannot be modified once initialized.
   First_Name : String (1 .. Max_Len);
   Last_Name  : String (1 .. Max_Len);
end record;
--  Person is an indefinite type (like an array)
```

Records with variant

```
type Node_Acc is access Node;

type Op_Kind is (Bin_Op, Un_Op);
-- A regular enum

type Node (Op : Op_Kind) is record
--        ^ The discriminant is an enum
   Id : Natural;
   case Op is
      when Un_Op =>
         Operand : Node_Acc;
      when Bin_Op =>
         Left, Right : Node_Acc;
      -- Those fields only exist when Op is Bin_Op
   end case;
   -- Variant part. Only one, at the end of the record
   -- definition
end record;
```

Quizz

Quizz 1: Is there a compilation error?

```
Buf : String (1 .. 10);
...
Buf (2 .. 4) := "Ab";
```

Quizz 2: Is there a compilation error?

```ada
type Person (Max_Len : Natural) is record
   First_Name : String (1 .. Max_Len);
   Last_Name  : String (1 .. Max_Len);
end record;

A : Person;
```

```
type Person (Max_Len : Natural) is record
   Name : String (1 .. Max_Len);
end record;

A : Person (20);
```

Quizz 4: Is there a compilation error?

```ada
type Person (Max_Len : Natural) is record
   Name : String (1 .. Max_Len);
end record;

A : Person := Person'(6, "Pierre");
```

Quizz 5: Is there a compilation error?

```ada
type Person (Max_Len : Natural) is record
   Name : String (1 .. Max_Len);
end record;

A : Person := Person'(20, "Pierre");
```

Quizz 6: Is there a compilation error?

```ada
type Date1_Acc is access Date;
type Date2_Acc is access Date;

D1 : Date1_Acc;
D2 : Date2_Acc;

D1 := D2;
```

Quizz 7: Is there a compilation error?

```
type Date_Acc is access Date;

D1 : Date_Acc := new Date;
D2 : Date_Acc;

D1 := D2;
```

Quizz 8: Is there a compilation error?

```ada
type String_Acc is access String;

S : String_Acc := new String'("Hello");
C : Character;

C := S.all (0);
```

```
type String_Acc is access String;
S : String_Acc := new String'("Hello");
C : Character;

C := S.all (1);
```

Quizz 10: Is there a compilation error?

```ada
type String_Acc is access String;
S : String_Acc := new String'("Hello");
C : Character;

C := S (1);
```

Quizz 11: Is there a compilation error?

```ada
type Node;
type Node_Acc is access Node;
type Op_Kind is (Bin_Op, Un_Op);

type Node (Op : Op_Kind) is record
   Id : Natural;
   case Op is
      when Un_Op =>
         Operand : Node_Acc;
      when Bin_Op =>
         Left, Right : Node_Acc;
   end case;
end record;

N : Node (Un_Op);
```

Quizz 12: Is there a compilation error?

```
type Node;
type Node_Acc is access Node;
type Op_Kind is (Bin_Op, Un_Op);

type Node (Op : Op_Kind) is record
   Id : Natural;
   case Op is
      when Un_Op =>
         Operand : Node_Acc;
      when Bin_Op =>
         Left, Right : Node_Acc;
   end case;
end record;

N : Node := (Un_Op, 2, null);
```

Quizz 13: Is there a compilation error?

```ada
type Node;
type Node_Acc is access Node;
type Op_Kind is (Bin_Op, Un_Op);

type Node (Op : Op_Kind) is record
   Id : Natural;
   case Op is
      when Un_Op =>
         Operand : Node_Acc;
      when Bin_Op =>
         Left, Right : Node_Acc;
   end case;
end record;

N : Node (Un_Op);

N.Left.Id := 12;
```

Quizz 14: Is there a compilation error?

```ada
type Node;
type Node_Acc is access Node;
type Op_Kind is (Bin_Op, Un_Op);

type Node (Op : Op_Kind) is record
   Id : Natural;
   case Op is
      when Un_Op =>
         Operand : Node_Acc;
      when Bin_Op =>
         Left, Right : Node_Acc;
   end case;
end record;

N : Node_Acc := ...

Put_Line (N.Left.Op'Image);
```

Privacy

Private part

```
package Stacks is
   procedure Hello;

private

   procedure Hello2;
   --  Not visible from external units
end Stacks;
```

Abstract data types: Declaration

```ada
package Stacks is
   type Stack is private;
   -- Declare a private type: You cannot depend on its
   -- implementation. You can only assign and test for
   -- equality.

   procedure Push (S : in out Stack; Val : Integer);
   procedure Pop (S : in out Stack; Val : out Integer);
private

   subtype Stack_Index is Natural range 1 .. 10;
   type Content_Type is array (Stack_Index) of Natural;

   type Stack is record
      Top : Stack_Index;
      Content : Content_Type;
   end record;
end Stacks;
```

Abstract data types: vocabulary

```ada
package Stacks is
   type Stack is private;
   -- Partial view

   procedure Push (S : in out Stack; Val : Integer);
   procedure Pop (S : in out Stack; Val : out Integer);
private

   subtype Stack_Index is Natural range 1 .. 10;
   type Content_Type is array (Stack_Index) of Natural;

   type Stack is record
      Top     : Stack_Index;
      Content : Content_Type;
   -- Full view
   end record;
end Stacks;
```

Abstract data types

```
-- No need to read the private part to use the package
package Stacks is
   type Stack is private;

   procedure Push (S : in out Stack; Val : Integer);
   procedure Pop (S : in out Stack; Val : out Integer);
private
   ...
end Stacks;
```

```
-- Example of use
with Stacks; use Stacks;

procedure Test_Stack is
   S : Stack;
   Res : Integer;
begin
   Push (S, 5);
   Push (S, 7);
   Pop (S, Res);
end Test_Stack;
```

Limited types

```ada
package Stacks is
   type Stack is limited private;
   -- Limited type. Cannot assign nor compare.

   procedure Push (S : in out Stack; Val : Integer);
   procedure Pop (S : in out Stack; Val : out Integer);
private
   subtype Stack_Index is Natural range 1 .. 10;
   type Content_Type is array (Stack_Index) of Natural;

   type Stack is limited record
      Top     : Stack_Index;
      Content : Content_Type;
   end record;
end Stacks;
```

Limited types

```
package Stacks is
   type Stack is limited private;
   ...
private
   type Stack is record  -- Full view is not limited
      ...
   end record;
end Stacks;
```

```
package Stacks is
   type Stack is limited private;
   ...
private
   ...
   type Stack is limited record -- Full view is limited
      ...
   end record;
end Stacks;
```