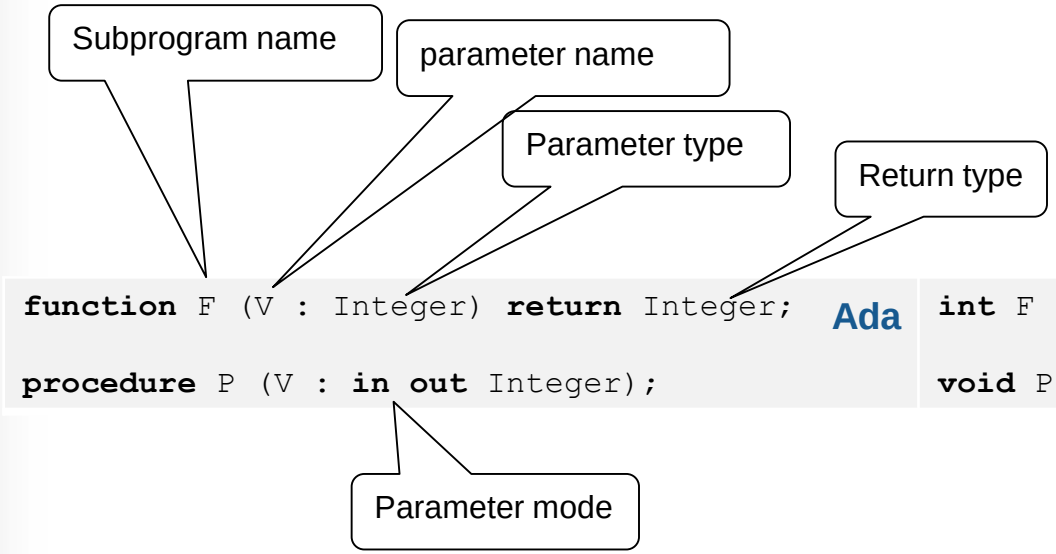
Subprograms

Presented by Quentin Ochem

university.adacore.com

Subprograms in Ada: Specifications



Ada

```
function F (V : Integer) return Integer;

procedure P (V : in out Integer);
```

C++

```
int F (int V);

void P (int *V);
```

- **Ada differentiates functions (returning values) and procedures (with no return values)**

Subprograms in Ada: Declaration and Body

Declaration

```
function F (V : Integer) return Integer;
```

Body

```
function F (V : Integer) return Integer is
   R : Integer := V * 2;
begin
   R := R * 2;
   return R - 1;
end F;
```

- **Declaration is optional, but must be given before use**
- **Functions result cannot be ignored**
- **Completion / body is introduced by “is”**

Parameter Modes

- **Mode "in"**
  - Specifies that actual parameter is not altered
  - Only reading of formals is allowed
  - Default mode
- **Mode "out"**
  - Actual is expected to be altered
  - Writing is expected, but reading is also allowed
  - *Initial value is not defined*
- **Mode "in out"**
  - Actual is expected to be both read and altered
  - Both reading & updating of formals is allowed

```
function F (V : in out Integer) return Integer is
   R : Integer := V * 2;
begin
   V := 0;
   R := R * 2;
   return R - 1;
end F;
```

Parameter Passing Mechanisms

- **Passed either “by-copy” or “by-reference”**
- **By-Copy**
  - The formal denotes a separate object from the actual
  - A copy of the actual is placed into the formal before the call
  - A copy of the formal is placed back into the actual after the call
- **By-Reference**
  - The formal denotes a view of the actual
  - Reads and updates to the formal directly affect the actual
- **Parameter types control mechanism selection**
  - Not the parameter modes

Standardized Parameter Passing Rules

- **By-Copy types**
  - Scalar types
  - Access types
  - Private types that are fully defined as by-copy types
- **By-Reference types**
  - Tagged types
  - Task types and Protected types
  - Limited types
  - Composite types with by-reference component types
  - Private types that are fully defined as by-reference types
- **Implementation-defined types**
  - Array types containing only by-copy components
  - Non-limited record types containing only by-copy components
  - Implementation chooses most efficient method

Subprogram Calls

- **If no parameter is given, no parenthesis is allowed**

```
function F return Integer;

V : Integer := F;
```

- **Named argument is possible**

```
procedure P (A, B, C : Integer);

P (B => 0, C => 0, A => 1);
```

- ***out* and *in out* modes require an object**

```
procedure P (X : out Integer);

V : Integer;

P (V);
```

Default Values

- **“in” parameters can be provided with a default value**

```
procedure P (A : Integer := 0; B : Integer := 0);
```

- **Default values are dynamic expressions, evaluated a the point of call if no explicit expression is given**

```
P;           -- A = 0, B = 0;
P (1);       -- A = 1, B = 0;
P (B => 2); -- A = 0, B = 2;
P (1, 2);    -- A = 1, B = 2;
```

Indefinite Parameters and Return Types

- **Subprograms can have indefinite parameters and return types**

```
function Comment (Stmt : String) return String is
begin
   return "/*" & Stmt & "*/";
end Comment;

S : String := Comment ("a=0"); -- return /*a=0*/
```

- **Constraints are computed at the point of call**
- **Don't assume boundaries!**

```
   procedure Init (Stmt : in out String) is
   begin
      for J in 1 .. Stmt'Length loop -- Incorrect
         Stmt (J) := ' ';
      end loop;
   end Init;

   S : String := "ABCxxx";
begin
   Init (S (4 .. 6));
```

Aliasing

- **Ada has to detect "obvious" aliasing errors**

```
   function Change (X, Y : in out Integer) return Integer is
   begin
      X := X * 2;
      Y := Y * 4;

      return X + Y;
   end;

   One : Integer := 2;
   Two : Integer := 4;

begin

   Two := Change (One, One);
   --  warning: writable actual for "X" overlaps with actual for "Y"

   Two := Change (One, Two) - Change (One, Two);
   --  warning: result may differ if evaluated after other actual in expression
```

Overloading (1/2)

- **Ada allows overloading of subprograms**

```
procedure Print (V : Integer);
procedure Print (V : Float);
```

- **Overloading is allowed if specification differ by**
  - Number of parameters
  - Type of parameters
  - Result type

```
subtype Positive is Integer range 1 .. Integer'Last;
procedure Print (V : Integer);
procedure Print (W : out Positive); -- NOK
```

- **Some aspects of the specification are not taken into account**
  - Parameter names
  - Parameter subtypes
  - Parameter modes
  - Parameter default expressions

Overloading (2/2)

- **Overloading may introduce ambiguities at call time**
- **Ambiguities can be solved with additional information**

```
   type Apples is new Integer;
   type Oranges is new Integer;


   procedure Print (Nb_Apples : Apples);
   procedure Print (Nb_Oranges : Oranges);

   N_A : Apples := 0;

begin

   Print (N_A);              -- OK
   Print (0);                -- NOK
   Print (Oranges'(0));      -- OK
   Print (Nb_Oranges => 0); -- OK
```

Operator Overloading

- **Default operators (=, /=, *, /, +, -, >, <, >=, <=, and, or…) can be overloaded, added or removed for types**

```
type Distance is new Float;
type Surface is new Float;

function "*" (L, R : Distance) return Distance is abstract; -- removes "*"
function "*" (L, R : Surface) return Surface is abstract;   -- removes "*"
function "*" (L, R : Distance) return Surface;              -- adds "*"

type Rec is record
   Unimportant_Field : Integer;
   Important_Field : Integer;
end record;

function "=" (L, R : Rec) return Boolean is
begin
   return L.Important_Field = R.Important_Field;
end "=";
```

- **“=“ overloading will automatically generate the corresponding “/=“**

Nested Subprogram and Access to Globals

- **A subprogram can be nested in any scope**

- **A nested subprogram will have access to the parent subprogram parameters, and variables declared before**

```
procedure P (V : Integer) is
   W : Integer;

   procedure Nested is
   begin
      W := V + 1;
   end Nested;
begin
   W := 0;
   Nested;
```

Null procedures and expression functions

- **A subprogram with no body can be declared as null (see later for actual usages of this)**

```
procedure P (V : Integer) is null;
```

- **A function that consists only of the evaluation of an expression can be completed at the specification level as an expression-function**

```
function Add (L, R : Integer) return Integer is (L + R);
```

- **This comes in handy when writing e.g. Pre / Post conditions**

Quiz

Is this correct? (1/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
   function F (V : Integer) return Integer is
   begin
      Put_Line (Integer'Image (V));
      return V + 1;
   end F;

begin

   F (999);
```

Is this correct? (1/10)

NO

```
function F (V : Integer) return Integer is
begin
   Put_Line (Integer'Image (V));
   return V + 1;
end F;

begin

   F (999);
```

F is called, but its return value is not stored

Is this correct? (2/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
procedure P (V : Integer) is
begin
   V := V + 1;
end P;
```

Is this correct? (2/10)

NO

```
procedure P (V : Integer) is
begin
   V := V + 1;
end P;
```

Compilation error, V is an input, read-only

Is this correct? (3/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
function F () return Integer is
begin
   return 0;
end F;

V : Integer := F ();
```

Is this correct? (3/10)

NO

```
function F () return Integer is
begin
   return 0;
end F;

V : Integer := F ();
```

Compilation error, parenthesis are not allowed if no parameters

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
procedure P (V : Integer) is
   procedure Nested is
   begin
      W := V + 1;
   end Nested;

   W : Integer;
begin
   W := 0;
   Nested;
```

Is this correct? (4/10)

NO

```
procedure P (V : Integer) is
   procedure Nested is
   begin
      W := V + 1;
   end Nested;

   W : Integer;
begin
   W := 0;
   Nested;
```

Compilation error, W is not yet visible at this point

Is this correct? (5/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
function F return String is
begin
   return "A STRING";
end F;

V : String (1 .. 2) := F;
```

Is this correct? (5/10)

NO

```
function F return String is
begin
   return "A STRING";
end F;

V : String (1 .. 2) := F;
```

Run-Time error,
the size of V and the value return by F do not match

Is this correct? (6/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
procedure P (Data : Integer);
procedure P (Result : out Integer);
```

Is this correct? (6/10)

NO

```
procedure P (Data : Integer);
procedure P (Result : out Integer);
```

Parameter names and modes are not significant enough for overload

Is this correct? (7/10)

YES
(click on the check icon)

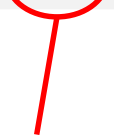
NO
(click on the error location(s))

```
   procedure P (V : Integer := 0);
   procedure P (V : Float := 0.0);
begin
   P;
```

Is this correct? (7/10)

NO

```
   procedure P (V : Integer := 0);
   procedure P (V : Float := 0.0);
begin
   P;
```



Compilation error, call to P is ambiguous

Is this correct? (8/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
   procedure Multiply
      (R : out Integer; V : Integer; Times : Integer)
   is
   begin
      for J in 1 .. Times loop
         R := R + V;
      end loop;
   end Multiply;

   Res : Integer := 0;
   X : Integer := 10;
begin
   Multiply (Res, X, 50);
```

Is this correct? (8/10)

NO

```
   procedure Multiply
      (R : out Integer; V : Integer; Times : Integer)
   is
   begin
      for J in 1 .. Times loop
         R := R + V;
      end loop;
   end Multiply;

   Res : Integer := 0;
   X : Integer := 10;
begin
   Multiply (Res, X, 50);
```

Erroneous, R is not initialized when entering Multiply

Is this correct? (9/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
   type My_Int is new Integer;

   function "=" (L, R : My_Int) return Boolean;

   function "=" (L, R : My_Int) return Boolean is
   begin
      if L < 0 or else R < 0 then
         return True;
      else
         return L = R;
      end if;
   end "=";

   V, W : My_Int := 1;
begin
   if V = W then
...
```

Is this correct? (9/10)

NO

```
   type My_Int is new Integer;

   function "=" (L, R : My_Int) return Boolean;

   function "=" (L, R : My_Int) return Boolean is
   begin
      if L < 0 or else R < 0 then
         return True;
      else
         return L = R;
      end if;
   end "=";

   V, W : My_Int := 1;
begin
   if V = W then
...
```

Infinite recursion on "="
A conversion could fix the problem, e.g. Integer (L) = Integer (R)

Is this correct? (10/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
   type Rec is record
      A, B : Integer;
   end record;

   function "=" (L : Rec; I : Float) return Boolean;

   function "=" (L : Rec; I : Float) return Boolean is
...

   A : Rec;
begin
   if A /= 0.0 then
   ...
```

Is this correct? (10/10)


```
   type Rec is record
      A, B : Integer;
   end record;

   function "=" (L : Rec; I : Float) return Boolean;

   function "=" (L : Rec; I : Float) return Boolean is
...

   A : Rec;
begin
   if A /= 0.0 then
   ...
```

OK – the declaration of "=" creates an implicit "/="

university.adacore.com