Introduction to Computers and Programming

Prof. I. K. Lundqvist

Lecture 1
Mar 11 2004

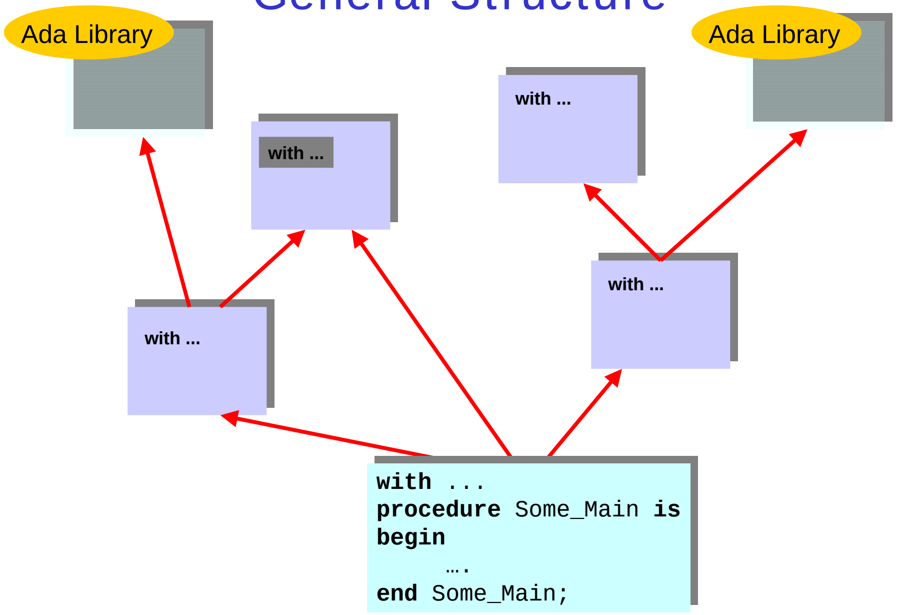
A Simple Ada Program

```
------------------------------------
-- Program Name: Hello
-- Purpose     : Display Hello World on screen
-- Programmer  : Joe B
-- Date        : March 11, 2004
------------------------------------

with Ada.Text_Io;
use Ada.Text_Io ;

procedure Hello is
begin
   Put("Hello World");
end Hello;
```

Typical Errors

- Compilation errors
  - Syntax
  - Semantics
- Run-time errors
  - Exception
- Logic or Algorithm errors
- Propagation Errors

General Structure



```
with ...
procedure Some_Main is
begin
     ….
end Some_Main;
```

Visibility Rules

- The ***visibility rules*** determine which declarations are visible and directly visible at each place within a program. The visibility rules apply to both explicit and implicit declarations

- Direct Visibility
  - ***immediate visibility***
  - ***use-visibility***

```
function Fact (N : Integer) return Integer is
begin
   if N <= 1 then
        return 1;
   else
        return N * Fact (N-1);
   end if;
end Fact;
```

fact.adb

```
with Fact;
procedure Main is
  procedure Hello is
  begin
    Ada.Text_Io.Put(“Hello”);
  end Hello;
begin
   for I in 1 .. Fact (4) loop
       Hello;
   end loop;
end Main;
```

main.adb

For Loop

```
for <loop_control_variable> in <lower_bound>..<upper_bound> loop
  <loop_body>
end loop;
```

- <loop_control_variable>
  - This is the name of the "variable that controls the loop". The loop control variable is incremented by one each time through the loop.
- <lower_bound>
  - The initial value given to the loop control variable.
- <upper_bound>
  - The final value of the loop control variable. The loop body executes one more time when the loop control variable = upper bound, then the loop terminates.
- <loop_body>
  - The code that's executed each time through the loop.

Loop Demo [CQ1]

```
procedure Loop_Demo is
  -- Square_Integer : Integer;
begin
  for Square_Integer in 1 .. 5 loop
      -- What does this do?
      Put (Item=>Square_Integer**2,
           Width => 3);
      New_Line;
     end loop;
end Loop_Demo;
```

The Program Output Appears As A, B or C ? [CQ1]

A	B	C
1	0001	1
4	0004	2
9	0009	6
16	00016	8
25	00025	10

Concept Question 1
Single Loop

1. The program output appears as **A**
2. The program output appears as **B**
3. The program output appears as **C**
4. I still don’t understand loops …

Nested Loops [CQ2]

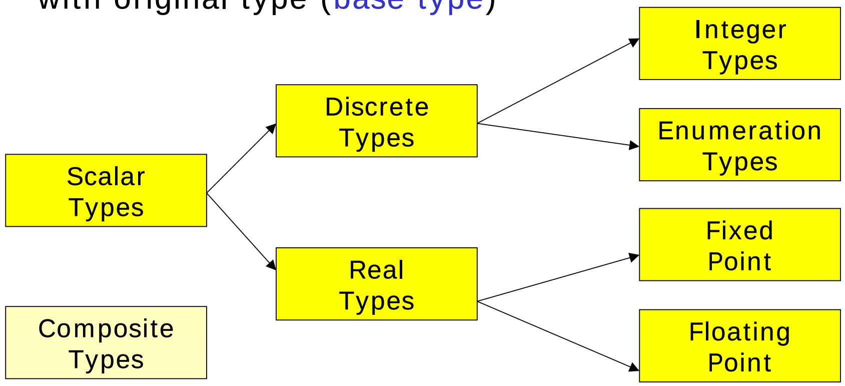
```
procedure Nested_Loop_Demo is
begin
   -- Outer loop
   for I in 2 .. 3 loop
      -- Inner loop
      for J in 1 .. 9 loop
         Put (I);
         Put (" *");
         Put (J);
         Put (" = ");
         Put (I*J);
         New_Line;
      end loop;
   end loop;
end Nested_Loop_Demo;
```

Concept Question 2
Nested Loops

1. The program goes through **2** iterations
2. The program goes through **9** iterations
3. The program goes through **18** iterations
4. The program goes through **27** iterations
5. I still don’t understand nested loops

Types

- Type
  - Set of Values
  - Set of Operations
- Subtype
  - Defines a subset of the values associated with original type (base type)



Type Attributes

- TYPE’First, TYPE’Last , TYPE’Image (X)

```
with Ada.Text_IO;
procedure Print (A : Integer; P : Float) is

    type My_Int is range -100 .. 1_000_000;
    T : My_Int := My_Int’Last;

    type Day is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
    D : Day := Day’First;
    B : Integer := Integer’First;

begin
   Ada.Text_IO.Put (Integer’Image (A));
   Ada.Text_IO.Put (Float’Image (P));
end Print;
```

```
function Compute (P, Q : Integer) return Integer is
   type My_Int is range -100 .. 1_000_000;
   T : My_Int;
begin
   T := P + 1;
   return (T + Q);
end Compute;
```

Will this compile?

Enumerations

- An enumeration type is a sequence of ordered enumeration literals:

```
type State is (Off, Powering_Up, On);
```

- No arithmetic defined

```
S1, S2 : State;
S1 := Off;
S2 := Powering_Up;
S1 := S1 + S2;   -- Illegal
```

- Can add/subtract one

```
State’Pred (S1)
State’Succ (S2)
```

Functions

```
<function_header>
   <local_variables_and_constants>

begin
   <function_body>
end <function_name>;
```

- <function_header>
  - – contains the function name and parameters.
- <local_variables>
  - – variables used in the function (but nowhere else).
- <function_body>
  - – the code the function executes.
- <function_name>
  - – the name of the function.

Function Header

```
function <function name> (
 <formal parameter name> : <data type>;
 <formal parameter name> : <data type>;
. . . ) return <data type> is
```

```
function Fact (N : Integer) return Integer is
begin
     if N <= 1 then
            return 1;
     else
            return N * Fact (N-1);
     end if;
end Fact;
```

Procedures

```
<procedure_header>
   <local_variables_and_constants>
begin
   <procedure_body>
end <procedure_name>;
```

- <procedure_header>
  - – contains the procedure name and parameters.
- <local_variables>
  - – variables used in the procedure (but nowhere else).
- <procedure_body>
  - – the code the procedure executes.
- <procedure_name>
  - – the name of the procedure.

Procedure Header

No Information Flow (No Parameters)

```
procedure <procedure name> is
```

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
begin
     Put_Line (“Hello”);
end Hello;
```

With Information Flow (With Parameters)

```
procedure <procedure name> (
   <formal parameter name> : <mode> <data type>;
   <formal parameter name> : <mode> <data type>;
   . . . ) is
```

```
with Ada.Text_IO;
procedure Increment (X : in out Integer;
                     Y : in out float) is
begin
   x:= x + 1; y := y + 1.4;
end Hello;
```

Procedure Calls

No Parameters

```
<procedure name>;
```

```
with Hello;
procedure Main is
begin
   Hello;
end Main;
```

With Parameters

```
<procedure name> (
   <formal parameter name> => <actual parameter name>;
   <formal parameter name> => <actual parameter name>,
  . . . );
```

```
with Increment;
procedure Main is
my_x : integer := 1;
my_y : float   := 2.0;
begin
   Increment(my_x, my_y);
end Main;
```

Arrays

```
type int_8_array  is array (1 .. 8) of Integer;
type CUBE6 is array (1..6, 1..6, 1..6) of Integer;
```

- Access elements using Indices
  - Single Dimension arrays A(I)
  - Two dimensional arrays A(I,J)
  - N dimensional array $A(i_1, i_2,..,i_n)$
- Loops can be used to access elements.

```
for I in 1 .. N loop
   for J in 1 .. N loop
      Put (B(I,J));
   end loop;
end loop;
```

Records

```
type My_Type_Record is
record
   my_boolean : Boolean;
   my_integer : Integer;
   my_real    : Float;
end record;
```

```
type My_Other_Type_Record is
record
   my_integer : Integer;
   my_real    : Float;
   my_boolean : Boolean;
end record;
```

```
Rec1 : my_type_record;
Rec2 : my_other_type_record;
```

- **Rec2 := Rec1;**
- **Rec1.my_boolean := Rec2.my_boolean;**
  Rec1.my_integer := Rec2.my_integer;
  Rec1.my_Real   := Rec2.my_real;

JK

- CP Review Session (NOT required)
  - 7:30