Lecture C1: Ada overview

Response to 'Muddiest Part of the Lecture Cards'

(17 respondents out of 60)

1) *I still don't understand nested loops* (1 student)
Statements within a **loop statement** can be built up of arbitrary statements, thus there may well be one **loop statement** within another. This is a very common construct to use in programs.
A small example: We want to write a triangle of stars (*) on the screen. The first row will have 1 star, the second row will have 2 stars, …, and the Nth row will have N stars. N will be read as input from the terminal. So, if N = 4 then the output will be

```
*
**
***
****
```

Algorithm:
1. Read in number N
2. Repeat the following step for each number K,from 1 to N.
2.1 print a row of K stars

Step 1:

```
Put_Line ("enter the number of rows to be printed");
Get(N);
```

Step 2:

```
for K in 1..N loop
-- 2.1 print a row of K stars
end loop;
```

Step 2.1

```
for J in 1..K loop
   Put('*');
end loop;
New_Line;
```

All put together:

```
with Ada.Text_Io, ada.Integer_Text_IO;
use Ada.Text_Io, ada.Integer_Text_IO;

procedure Nested_Loops is
   N : integer;
begin
   Put_Line ("enter the number of rows to be printed");
   Get(N);
   for K in 1..N loop
      for J in 1..K loop
         Put('*');
      end loop;
      New_Line;
   end loop;
end Nested_Loops;
```

Exercise: if we instead wanted the following output:

```
****
***
**
*
```

what should have needed to be changed in the code? (Hint: only one word needs to be added…)

2) *Isn't declaring* `My_Counter : integer := 10;` *at the beginning of a procedure and also using* `My_Counter` *in a for loop*

```
      for My_Counter in 1..5 loop
```

in the same procedure bad Ada Style (i.e. frowned upon)? (1 student)

Exactly, the declaration does violate the style guide. However, the purpose of the example was to demonstrate the change in scope within the **for loop** construct.

According to "The Ada Style Guide" one should (amongst many other things)

> "Limit the scope of a renaming declaration to the minimum necessary scope"

Information about loops and visibility can be found via the following links:
http://www.iste.uni-stuttgart.de/ps/ada-doc/style_guide/sec_5b.html#5.6.4
http://www.iste.uni-stuttgart.de/ps/ada-doc/style_guide/sec_5b.html#5.10

3) *What was the result of the code at the end of the notes*? (1 student)
After removing the syntax and semantic error, but keeping the logical/propagation error in the file, the result of test-running the code was printing the following on the screen:

```
This should never happen
```

4) *Didn't hear the answer to the difference between visibility types.* (1 student)
Taken from the LRM (Language Reference Manual), section 8

"There are two kinds of direct visibility: ***immediate visibility*** and ***use-visibility***. A declaration is **immediately visible** at a place if it is directly visible because the place is within its immediate scope. A declaration is **use-visible** if it is directly visible because of a use_clause. Both conditions can apply."

In the example on slide 6, the procedure `Hello` is immediately-visible, and the procedure `Fact` was use-visible.

5) *In ConceptQuestion_2, the program only goes through 1 iteration. The loops repeat 18 times.* (1 student)
You are correct. The question should have been the loops iterate X times. The program only iterates once

6) *Don't understand subset example* (1 student)
I assume you mean the following from slide 13 on Types?
"Subtype: Defines a subset of the values associated with original type (base type)."

The declaration of a subtype does not mean that a new type has been created. It simply means that a name has been introduced for a subset of a base type. For example, there are two predefined subtypes in Ada.

```
   subtype Natural is Integer range 0..Integer'Last;
   subtype Positive is Integer range 1..Integer'Last;
```

Each of those subtypes defines a subset of the values its base type Integer has. All the usual Integer operations remain available. A Positive integer is still an integer.

You can also define your own types as in

```
   type Day_Of_Week is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
   subtype Workday is Day_Of_Week range Mon..Fri;
```

7)
```
with Ada.Text_Io;
procedure Start_Cp_Class is
   Professor is constant = "Kristina";
   Stay_Awake     : Boolean := True;
   Come_To_Class : Boolean := True;
   No_Of_Classes : Integer;
begin -- start term
   Ada.Text_Io.Put (Item => "Welcome");
end Start_Cp_Class;
```

This code will generate a **compilation error**, but if the third line instead would have been

```
Professor : constant String := "Kristina";
```

This code would no longer generate a **compilation error**, but the compiler would still give us four **warnings**. ;-)

8) *No mud* (10 students)