Lecture C8 and C9: Types / Packages

Response to 'Muddiest Part of the Lecture Cards'

(41 respondents, out of 70students)

1) The difference between functions and procedures? (7 students)

Both functions and procedures are classified as subprograms in Ada. The overall format of both is the same. The difference however is in the details of the header and the number of values returned.

Functions have the format:

```
<function header>    <local variables and constants> begin  <function
body> end <function name>;
```

The function header has the following format:

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

Functions must always return a value of the type specified in the function header. In addition, all the parameters of the function header are of type **in**.

Procedures on the other hand can have zero or more outputs. The parameters in the procedure header can be of type **in**, **out** or **in out**.

Procedures have the format:

```
<procedure   header>         <local   variables   and   constants> begin
      <procedure body> end <procedure name>;
```

The procedure header has the following format:

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

Mode specifiers are:

- **in** : the variable is of type input and cannot be changed inside the subprogram. In other words, it can only appear in an expression or the right hand side of an assignment statement
- **out** : the variable is an output variable and can only appear in the left hand side of an assignment statement.
- **in out :** the variable can appear on either side of an assignment statement or in an expression.

2) I do not understand Formal and Actual paramters. Please explain a transfer of control? (and similar questions) (9 students)

The fundamental idea behind using parameters in function or procedure calls is to exchange information. Ada uses a mechanism called *pass by value*, to exchange information between subprograms.

Consider the procedure call below:

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

In this case, the calling subprogram (the procedure or function in which this above statement is present) is passing information to the procedure it is calling. The `formal parameter name` is the name present in the subprogram header and the `actual parameter name` is the name of the variable in the calling subprogram. The advantage of using this method of invocation, is that the programmer does not have to remember the exact order in which the variables were declared in the subprogram header. If the `formal parameter name =>` construct `actual parameter name` is not used, the variables are filled from left to right and may result in either a type mismatch or in the wrong values being used for computation.

Consider the example below:

```
with Ada.Text_Io;
use Ada.Text_Io;

with Ada.Integer_Text_Io;
use Ada.Integer_Text_Io;

procedure Callee (
    X : in     Integer;
    Y :    out Integer;
    Z : in out Integer  ) is
begin

  Y:= X+Z;
  Z:= X-Z;
end Callee;
```

```
procedure Caller is
  A,
  B,
  C : Integer;


begin
  A:= 2;
  C:= 3;
  Callee(A,B,C);
  Put(A);
  New_Line;
  Put(B);
  New_Line;
  Put(C);
  New_Line;

  Callee(C,B,A);
  Put(A);
  New_Line;
  Put(B);
  New_Line;
  Put(C);
  New_Line;

end Caller;
```

When the first procedure call Callee(A,B,C) occurs,

```
X:= A; -- X is an in variable
Z:= C; -- Z is an in out variable
```

The procedure callee executes, at the end of the execution,

```
B:= Y; -- Y is an out variable
C:= Z; -- Z is an in out variable
```

The output is 2,5,-1

When the second procedure call Callee(C, B, A) takes places,

```
X:= C; -- X is an in variable
Z:= A; -- Z is an in out variable
```

The procedure callee executes, at the end of the execution,

```
B:= Y; -- Y is an out variable
A:= Z; -- Z is an in out variable
```

The output is –3, 1, -1

If you change the second procedure call to Callee(Z=> C, y => B, x => A);then the output of the second call (assuming the first call is the same) is 2,1,3. Note that the parameters are passed in the same way as the first call.

3) Difference bwetween Sub-types, enumeration-types and derived types? (7 students)

Subtypes define a range of values within the parent type. In addition, they inherit the properties of the parent type and conversions between a subtype and its parent type are allowed.
For instance:

```
procedure Type_Illustrator is

  subtype Small_Number is Integer range 1 .. 20;
  X : Integer;
  Y : Small_Number;

begin

  X:= 10;
  Y:= 15;

  X := X+Y;
  Y:= X+Y;

end Type_Illustrator;
```

The program compiles correctly but the y:= x+y statement will raise a constraint error because the value to be assigned to Y is 35, which is out of the range of values specified for the subtype.

A **derived type** is a type whose characteristics are derived from the parent type. For example:

type Counter **is new** Positive;

All the properties and operations of the parent are defined for the child and can be redefined by the user. They derived type however is a unique type and cannot be mixed with the type of the parent (unlike subtypes).

An **Enumeration** is a sequence of ordered literals. For example:

type Color **is** (White, Red, Yellow, Green, Blue, Brown, Black);

A variable of type Color can only take values White, Red, Yellow, Green, Blue, Brown, or Black. The only predefined operations on enumerations are assignment, equality, Color'Pred, Color'Succ, Color'Pos, Color'Val, Color'Image, Color'Value. To define other operations on enumeration types (such as get and put), the programmer has to define a new package (discussed in detail in the next question).

4) Enumeration_IO, Color_IO, what does it mean? (2 students)

The operations for input and output are not defined for the enumeration type Color. Ada has a **generic package** called Ada.Text_IO.Enumeration_IO which allows the user to create the input and output operations for Color automatically using the construct shown below:

```
PACKAGE Color_IO IS
   NEW Ada.Text_IO.Enumeration_IO(Enum => Color);
```

The construct Enum => Color allows the input/ output routines to check if the value to be obtained or displayed is within the legal range of values or not.

5) How does a computer deal with something like taking the maximum values of something? (1 students)

The number of bits that can be used to obtain a number are fixed for each pre-specified type. The computer can derive the maximum value that can be stored in each type depending on the number of bits used to store the value and the representation scheme used (think about floating point notation).

6) Can we work through more examples of conversions [ASCII, binary]? (2 students)

Take a look at the Number Systems handout. If you need to work out more examples, we will work out some in the recitation sessions and in the review session before the exam.

7) Is there a way to memorize everything? (1 students)

The concepts presented in your lectures and readings are fundamental. You should remember all the algorithms (converting from binary to decimal, 2's complement, floating point etc.). As far as Ada goes, you have been given a summary of Ada syntax, which should help in understanding and remembering. You are not expected to be perfect in the syntax (the compiler will help catch syntax errors). You do however need to understand the semantics (the meaning underlying the syntax).

8) Please provide pre-readings for the lectures? (1 students)

All the readings for each lecture are posted on the course website.

9) How much syntax is on the test? (1 students)

The quiz will test your understanding of Ada syntax and your ability to write Ada programs. While points will be taken off for incorrect syntax (for example: using the syntax of another language), you will not lost points for something like forgetting a ';'.

10) How detailed an algorithm do you want? (1 students)

An algorithm should capture your thought process. The algorithm details the steps you need to take to solve the problem. It should not be an English version of your code!

11) On the 4th line of the 'bus example' max_seated : constant no_on_buses := 50; shouldn't no_on_buses be max_seated? (1 students)
The slide is correct if you take into concideration that 30 people could be standing up on the bus.

12) "No mud" (7 students)
Good =)