Lecture C12

Response to 'Muddiest Part of the Lecture Cards'

(6 respondents)

1) *From the Dates example, it seems that the input can either be uppercase or lower case, but I did not see any character conversion (to all uppercase or to all lower caser). Does this mean that upper case and lower case letter are worth the same in Ada*?

```
type Days is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
package Day_Io is new Enumeration_Io (Days); use Day_Io;
```

The program (example 2), is working with days as enumerations types not character types. The key thing to understand here is the get function implemented using Day_IO is case insensitive. See A.10.10 in the Language Reference Manual for more details. If the input was character input, then there is a significant difference between upper_case and lower_case characters.

2) *What is the purpose of a raise statement? I still don't understand how this is useful. Would you ever use it outside of an exception handler*?
The raise statement is used to identify the occurrence of an exception. If there are user defined exceptions in the program, then it is left to the programmer to identify it and transfer control to the exception handler.

For example, you can define a variable of type exception, which the programmer can use to identify the occurrence of an overflow.

```
        Overflow : exception;
```

You can check for the occurrence of an overflow in the program and raise the exception to transfer control to the exception handler using:

```
        raise  Overflow;
```

The programmer can also raise predefined exceptions in the same manner.

When an exception is raised it means that the code segment/block (the segment starts with **begin** and ends with **end**) that was executed is abandoned and we jump to a matching exception handler. If no exception handler is found, the current subprogram is left and we look for an exception handler in the caller. If once again no exception handler is found, the process is repeated and we exit that subprogram and looks in the caller, the process is repeated until we find a matching exception handler or until we exit the complete program.

Note: the exception handler has to be placed just before the **end** statement of the block.
Ex:

```
     function Tan (X : Float )
          return Float is
        begin
           return Sin(X) / Cos(X);
        exception
           when Constraint_Error =>
              if (Sin(X)>=0.0 and Cos(X)>= 0.0) or
                    (Sin(X)< 0.0 and Cos(X)< 0.0)
                    then
                 return Float'Last;
              else
                 return -Float'Last;
              end if;
        end Tan;
```

The raise statement can also be made inside the exception handler, to inform the calling program that an error occurred. A raise statement without a named exception can only be used *within* an exception handler. A raise statement without the exception name re-raises the exception being handled to the next level.

```
        function Tan (X : Float )
          return Float is
        begin
           return Sin(X) / Cos(X);
        exception
           when Constraint_Error =>
              Put_Line ("The value of tangent is too big");
              raise;
        end Tan;
```

The raise occurring here, informs the calling program that a constraint error occurred because the computation is incorrect.

3) *How do we know what different types of errors to expect*?

It is the programmer's responsibility to identify the errors. This occurs both by analyzing the system at design time and by testing the implemented code. For example: Boundary value testing is a good way of identifying exceptions. Most of the predefined exceptions in Ada, help the programmer catch common errors, however, as the program size grows, it is easier to define your own exceptions for the sake of clarity.

4) *How does it know whether to throw my exception, or another or XXX when handling them*?

The idea of user defined exceptions is that the programmer creates a check and identifies the occurrence of the exception before the run-time system. For example, in the code fragment shown below, the programmer identifies and raises the exception of dividing by zero. If this check were not in place, a constraint error would have been raised by default.

```
function Tan (X : Float )
  return Float is
  divide_by_zero : exception;
begin
  if (Cos(x) = 0) then
       raise divide_by_zero;
  end if;
  return Sin(X) / Cos(X);
exception
  when divide_by_zero =>
           Put_Line ("Trying to divide by 0");
            raise Constraint_Error;

  when Constraint_Error =>
      Put_Line ("The value of tangent is too big");
      raise;
end Tan;
```

5) *No mud*