16.070

Introduction to Computers & Programming

Ada V

Loops, writing procedures, exception handling

Guest Lecturer: Tony Bogner
Charles Stark Draper Laboratory

FOR Loops

- Ada LOOP statement allows for repetition of a sequence of statements
- Three forms of a LOOP statement

```
FOR loop_specification LOOP     WHILE condition LOOP     LOOP
   ...                              ...                    ...
END LOOP;                       END LOOP;               END LOOP;
```

- FOR loop is used when executing a specific number of iterations with a loop parameter taking in turn all values of a discrete range

```
DECLARE
  S, M : Integer := 0;
BEGIN
  Text_Io.Put("Enter Test Score:"); Integer_Text_Io.Get(M); S = S + M;
  Text_Io.Put("Enter Test Score:"); Integer_Text_Io.Get(M); S = S + M;
  Text_Io.Put("Enter Test Score:"); Integer_Text_Io.Get(M); S = S + M;
```

FOR Loop Implementation

```
  FOR N IN 1..3 LOOP
    Text_IO.Put("Enter Test Score:"); Integer_Text_Io.Get(M); S = S + M;
  END LOOP;
```


FOR Loops: Examples

```
FUNCTION Factorial (N: Positive) RETURN Positive IS
   M: Positive := 1;
BEGIN
  FOR I IN 1..N LOOP
     M := M * I;
  END LOOP;
  RETURN M;
END Factorial;
```

```
N := 3;
FOR I IN REVERSE 1..N LOOP
  ...
  Integer_Text_Io.Put(Item => I, Width => 5);
  Integer_Text_Io.Put(Item => N, Width => 5);
  Text_IO.New_Line;
  N := 10;
END LOOP;
Integer_Text_Io.Put(Item => N, Width => 5);
Text_IO.New_Line;
-- Sample output
   3     3
   2    10
   1    10
  10
```


Nested Loops

- Loops can be nested as in example below:

```
N := 3;
FOR I IN 1..N LOOP
  FOR J IN 1..I LOOP
    Text_Io.Put('*');
  END LOOP;
  Text_IO.New_Line;
END LOOP;

-- Sample output
*                    → I=1; J=1..1
**                   → I=2; J=1..2
***                  → I=3; J=1..3
```

Subtypes and FOR Loops

- `FOR` loops presented thus far have all had integers as their counter. Technically, `FOR` loop counters can be any discrete type
- `SUBTYPE` can be used to specify a range of values in a `FOR` loop
- Example:

```
TYPE Days IS (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
SUBTYPE Weekdays IS Days RANGE Mon .. Fri;

FOR Today IN Days LOOP             -- Loop through whole week
  ...
END LOOP;

FOR Today IN Weekdays LOOP         -- Loop through weekdays
  ...
END LOOP;

FOR Today IN REVERSE Weekdays LOOP -- Loop through weekdays in reverse
  ...
END LOOP;
```


Subtypes and IF Statements

- `SUBTYPE` can also be used to specify a range of values for a *membership* test within an `IF` statement
- Example:

```
DECLARE
  TYPE Subframe_Data IS (SF1, SF2, SF3, SF4, SF5);
  SUBTYPE Almanac_Data IS Subframe_Data RANGE SF4 .. SF5;
  Current_Subframe : Subframe_Data;
BEGIN

  ... -- Current_Subframe set by data from tracking loop

  IF Current_Subframe IN Almanac_Data THEN
    Store_Almanac_Data;
  END IF;
```


General LOOP Statement

- `FOR` loops are quite useful but have limitations:
  - Counter limited to discrete range
  - Count up or down by 1 ('SUCC or 'PRED)
- General `LOOP` statement can be used to repeat a sequence indefinitely
- Example:

cosh $x = 1 + x^2/2! + x^4/4! + x^6/6! + ...$

```
DECLARE
  Cosh_X, Term : Float := 1.0;
  X : Float := 2.0;
  I : Integer := 0;
BEGIN
  LOOP
    I := I + 2;           -- increment by 2
    Term := (Term * X ** 2) / Float((I) * (I-1));
    Cosh_X := Cosh_X + Term;
  END LOOP;
END
```


EXIT Statement

- Exiting the general loop requires an `EXIT` statement
- One approach

```
LOOP
  I := I + 2;              -- increment by 2
  Term := (Term * X ** 2) / Float((I) * (I-1));
  Cosh_X := Cosh_X + Term;
  IF Term < 1.0e-6 THEN EXIT; END IF;
END LOOP;
```

- Another approach is to replace `IF` with `WHEN`

```
EXIT WHEN (Term < 1.0e-6);
```

- Can exit loops when nested:

```
Outer:
LOOP
  Inner:
  LOOP
    ...
    EXIT Inner WHEN condition;
    ...
  END LOOP Inner;
  ...
END LOOP Outer;
```

```
Search:
FOR I IN 1..N LOOP
  FOR J IN 1..M LOOP
    IF condition THEN
      I_VALUE := I;
      J_VALUE := J;
      EXIT Search;
    END IF;
  END LOOP;
END LOOP Search;
```

WHILE Statement

- If `EXIT` condition appears at the top of the loop, consider a `WHILE` statement

```
WHILE Term >= 1.0e-6 LOOP
  I := I + 2;             -- increment by 2
  Term := (Term * X ** 2) / Float((I) * (I-1));
  Cosh_X := Cosh_X + Term;
END LOOP;
```

- Condition for `WHILE` statement is a compliment/opposite of that used in the `EXIT` statement

Procedures

- A procedure encapsulates an activity
- A procedure implements an algorithm
- Procedure is called as a statement
- Procedures differs from a function in that
  - Procedure starts with reserved word `PROCEDURE`
  - Procedure name must be an identifier
  - Procedure does not return a result
  - Parameters, if present, may be of three different modes:
    - **`in`**
    - **`out`**
    - **`in out`**
- Procedure specification tells you about the interface to the procedure
- Procedure body implements the algorithm

Procedure Parameter Modes

- An **`in`** mode parameter can not be modified in a procedure
  - Acts as a constant
  - Any attempt to modify an **`in`** parameter will be caught by compiler and flagged as an error
  - The value of an **`in`** parameter in a procedure call may be an expression or constant
- An **`out`** mode parameter is used for generated values
  - The value of the actual parameter in the procedure call does not matter
  - The actual parameter must be a variable
  - The parameter must be set before it is read in the procedure
- An **`in out`** mode parameter is used when a procedure will modify the parameter
  - The procedure may use the parameter assuming it has a valid value
  - The parameter may be assigned a new value

Procedure Syntax

```
PROCEDURE proc_name (formal_parameters);
```

Specification

```
PROCEDURE proc_name (formal_parameters) IS
  local declarations
BEGIN
  statement sequence
END proc_name;
```

Body

Example Procedure and Invocation

```
PROCEDURE Add(A, B : IN Integer; C : OUT Integer);

PROCEDURE Add(A, B : IN Integer; C : OUT Integer) IS
BEGIN
  C := A + B;
END Add;

P, Q : Integer;
...
Add(A => 2 + P, B => 37, C => Q);
Add(2+P, 37, Q);
```

```
PROCEDURE Increment(X : IN OUT INTEGER) IS
BEGIN
  X := X + 1;
END Increment;

I : Integer;
...
Increment(X => I);
```


Writing Packages

- Packages used to encapsulate
- Package used to control access to resources
- Specification gives the interface to the package
- Syntax:

```
PACKAGE pack_name IS
  list of specifications of resources
  provided by package
END pack_name;
```

Specification

```
PACKAGE BODY pack_name IS
  sequence of function and procedure bodies
  that implement the resources identified
  in the package specification
BEGIN
   sequence of initialization statements, if any
END pack_name;
```

Body

Example Package Specification and Body

```
PACKAGE Gps_Rcvr IS
  PROCEDURE Put_Rcvr_Ant(X, Y, Z : IN FLOAT);
  FUNCTION Is_Ant_Installed RETURN Boolean;
END Gps_Rcvr;
```

Specification

```
PACKAGE BODY Gps_Rcvr IS
  Loc_X, Loc_Y, Loc_Z : FLOAT := 0.0;
  Location_Set : Boolean := FALSE;
  PROCEDURE Put_Rcvr_Ant(X, Y, Z : IN FLOAT) IS
  BEGIN
    Location_Set := TRUE;
    Loc_X := X; Loc_Y := Y; Loc_Z := Z;
  END;
  FUNCTION Is_Ant_Installed RETURN Boolean IS
  BEGIN
     RETURN Location_Set;
  END Is_Installed;
END Gps_Rcvr;
```

Body

Using the Package

```
with Gps_Rcvr; -- specify packages we depend on
use Gps_Rcvr;

procedure Initialize is
begin
     -- If the receiver is not installed, then …
     if not Is_Ant_Installed then
       Put_Rcvr_Ant(3.0, 4.0, 5.0); -- In meters from c.g.
     end if;
end;
```

Exception Handling

- Exceptions are used to tell the programmer that an error condition has occurred
- Examples:
  - Value out of range of the type or subtype
  - I/O error (such as trying to read a non-numeric character as a number
  - Programmer defined exceptions
- Exception handlers allow the programmer to attempt to fix the problem and continue execution
- Exception handlers are associated with blocks

```
BEGIN
  sequence of statements  -- Only statements executed if no exception
EXCEPTION                 -- If exception occurs, one of following
                          -- handlers is executed
  WHEN exception_name1 =>      -- handler for exception_name1
    sequence of statements1....
  WHEN exception_nameN =>      -- handler for exception_nameN
    sequence of statementsN
END;
```


Predefined Exceptions

- Common, predefined exceptions include:
  - **`Constraint_Error`**

    Generally raised when storing a value in a variable that is out of range for the variable—value exceeds variable type or subtype
  - **`Program_Error`**

    Attempt to violate a control structure in some way such as running into the end of a function
  - **`Storage_Error`**

    Occurs when the program runs out of storage space, perhaps through a recursive invocation of a function or procedure
  - **`Ada.Text_IO.Data_Error`**

    Attempt to read a value which is invalid for the variable being read

Trivial/Bad Example: Exception Handler

- A trivial example:

```
TYPE Days IS (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

FUNCTION Tomorrow(Day : IN Days) RETURN Days IS
BEGIN
  RETURN Days'SUCC(Day);
EXCEPTION
  WHEN Constraint_Error =>
    RETURN Days'First;
END Tomorrow;
```

- Example code is valid Ada and will work as expected
- Example is bad because the occurrence of the exception is not a rare condition

Better Example: Robust Input

```
WITH Ada.Text_IO;
WITH Ada.Integer_Text_IO;
PROCEDURE Get_Time_From_User IS
  SUBTYPE Time_T IS Integer RANGE 0..60*60*24;
  User_Time : Time_T;
BEGIN
  LOOP
    BEGIN    -- start block for exception handler
      Ada.Text_IO.Put(Item => "Enter UTC time (seconds) of day between ");
      Ada.Integer_Text_IO.Put(Item => Time_T'First, Width => 0);
      Ada.Text_IO.Put(Item => "and ");
      Ada.Integer_Text_IO.Put(Item => Time_T'Last, Width => 0);
      Ada.Integer_Text_IO.Put(Item => ":");
      Ada.Integer_Text_IO.Get(Item => User_Time);
      EXIT
    EXCEPTION
      WHEN Constraint_Error =>
        Ada.Text_IO.Put_Line(Item => "Time out of range.  Try again");
      WHEN Ada.Text_IO.Data_Error =>
        Ada.Text_IO.Put_line(Item => "Value not an integer.  Try again");
    END      -- end block for exception handler
  END LOOP
END;
```

SAMPLE RUN

```
Enter UTC time (seconds) of day between 0 and 86400: -90
Time out of range.  Try again
Enter UTC time (seconds) of day between 0 and 86400: asd
Value not an integer.  Try again
Enter UTC time (seconds) of day between 0 and 86400: 2456
```