Introduction to Computers and Programming

Prof. I. K. Lundqvist

Reading: FK pp. 199-209, 220-223, 266-267

Lecture 11
Sept 29 2003

Recap

- Scope : determine which declarations are visible and directly visible at each place within a program. The visibility rules apply to both explicit and implicit declarations.
  - *immediate visibility* and *use-visibility*
-
```
case selector is
   when value_list_1 =>
      statement(s)_1;
   when value_list_2 =>
      statement(s)_2;
      …
   when others =>
      statement(s)_n;
end case;
```

Iteration

- Definite iteration
  - FOR statement
- Indefinite iteration
  - WHILE statement
  - General LOOP statement

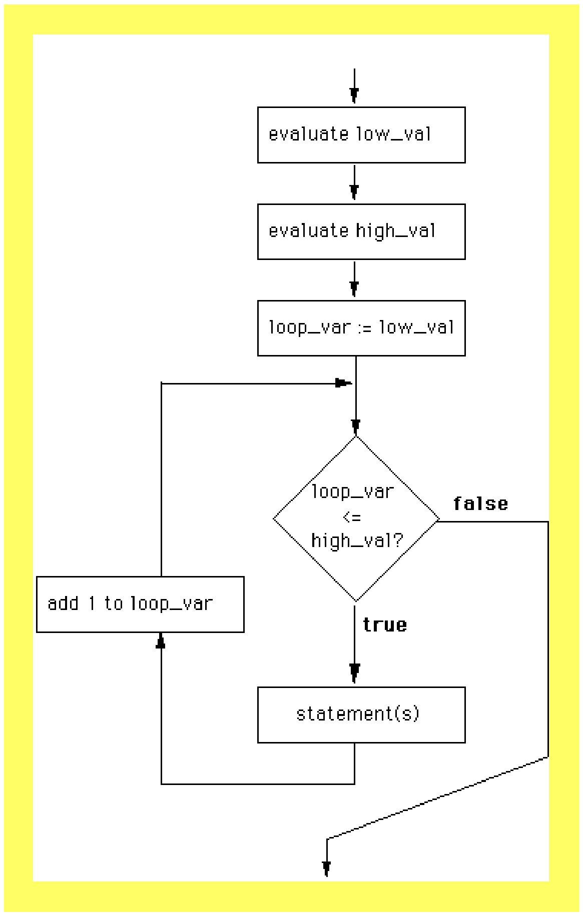
For Statement

-
  ```
  for loop_var in low_val .. high_val
  loop
     statement(s);
  end loop;
  ```

-
  ```
  for i in -1 .. 10 loop
     PUT(i); NEW_LINE;
  end loop;

  for i in 1 .. 10 loop
     PUT(i); NEW_LINE;
  end loop;

  for i in 2 .. n-1 loop
     PUT(i); NEW_LINE;
  end loop;
  ```



Courtesy of Chris Lokan. Used with permission.

[assignment_average.adb] [not covered in class]

- Specification
  - A program is required to calculate the average number of assignments marked by a lecturer per month over a 12 month period.

    The program will ask the user for the number of assignments marked each month, calculate the average number marked per month, and display this value.

[User interface]

- ASSIGNMENT AVERAGE PROGRAM

```
No. marked in month  1: xxx
No. marked in month  2: xxx
...
No. marked in month 12: xxx

Average per month is    yyy.yy
```

[Algorithm]

- Initialization
  - Display heading
  - Set total to zero
- Get values over year
  - For each month
    - Prompt for and get number marked
    - Add to total
- Calculate average
- Display average

[Data Design]

NAME	TYPE	Notes
max_month	(const) 12	No of months to process
assgn_month	Integer	No of assignments in a month
total_assgn	Integer	Total assignments in a year
this_month	Integer	Loop parameter
average_assgn	Float	Average assignment per month

WHILE Statement

-
  ```
  while test loop
     statement(s);
  end loop;
  ```
  Ex: **millionaire.adb**
- While loops may be designed as a repeat structure, to execute at least once

-
  ```
  j := -1;
  while (j < 0) loop
     put (“Enter positive j: );
     get (j); skip_line;
  end loop;
  ```

WHILE Statement

- A while loop may not execute at all

```
tot := 0;
PUT("Enter j (-1 to exit): ");
GET(j); SKIP_LINE;
while (j /= -1) loop
   tot := tot + j;
   PUT("Enter j (-1 to exit): ");
   GET(j); SKIP_LINE;
end loop;
PUT("Total is "); PUT(tot); NEW_LINE;
```

While example

- Specification
  - The user is to be prompted to enter 'Y' (or 'y') to indicate yes, or 'N' (or 'n') to indicate no. If either of these responses is provided, the program confirms it and terminates. Any other response results in the prompting and input being repeated.
- User interface
  - Please enter ‘Y’ for yes ‘N’ for no: X
    Please enter ‘Y’ for yes ‘N’ for no: N
    You entered No.

While example Algorithm

- Do initialization
  - Set response to ‘x’
- Get yes no
  - While response is neither ‘Y’ nor ‘N’
    - Prompt for and get response
    - Check input
      - Convert ‘y’ to ‘Y’
      - Convert ‘n’ to ‘N’
- Confirm input
  - If response is ‘Y’
    - Display yes confirmation
  - If response is ‘N’
    - Display no confirmation

Sentinel-controlled loops

- A sentinel value
  - A unique value that indicates end-of-data
  - It cannot be a value that could occur as data
  -

```
sentinel : constant := ???;
loop
   read (item);
   exit when item = sentinel;
   process the item;
end loop;
```

Flag-controlled loops

- A **flag** is a Boolean variable
  - A value of False indicates that the desired event has not yet occurred
  - True indicates that it has occurred
  -

```
flag : boolean;
flag := false;
while not flag loop
   do some processing;
   if desired event has happened then
      flag := True;
   end if;
   do some processing;
end loop;
```

Flag-controlled loops Example

```
DigitRead := FALSE;
while not DigitRead loop
   get (c);
   if (c >= ‘0’) and (c <= ‘9’) then
      DigitRead := TRUE;
   end if;
end loop;
```

WHILE vs. FOR

- The **while** loop is most general
  - Used for indefinite iteration
  - By setting value of variables used in the test, can be made to execute at least once, or maybe never
- **For** loops
  - Used for definite iteration (know number of times to run before loop)
  - Can be replaced by a **while** loop, but usually not as elegant

WHILE vs. General LOOP

- It is easier to reason with while loops than with general loops
- While
  - Obvious where loops exits
  - Can guarantee the entry test has been passed when executing the statements of the loop body
- General loop
  - Sometimes code is simpler
  - There is no guarantee when executing the statements before the test (this may be the first time through the loop)

Which loop statement to use?

- For **definite loops**, use for
  - Automated control
  - Tells reader that number of iterations known
- For **indefinite loops**, where pre-test is natural, use while
  - Simpler reasoning
- For **other indefinite loops**, use general loop
  - Less cluttered code
- Usually prefer for and while

Loop control

- There are three aspects to loop control for *any* loop:
  - initialization of loop parameter(s)
    - automatic in a for loop
    - *before* a while loop
    - *before* a general loop, or in the statements *before* the exit test
  - test
    - automatic in a for loop
    - while: test for *continuation*
    - general: test for *termination*
  - update loop parameter
    - not allowed inside for loop
    - *must* update inside indefinite loop

Loop design

- Choose the appropriate loop statement, and then specify the three aspects to loop control. Approaches to loop design:
  - analyze the individual application
  - recognize standard situations, use standard loops for those situations (eg sentinel-controlled loops, flag-controlled loops)
  - use loop templates
  - definite or indefinite?
  - when to test for termination?
  - what is appropriate initialization etc?

Nested loops

- When want to do something multiple times, and then do that multiple times.
  - Printing a 2-dimensional table
  - Printing the average mark of several exams
- Programming process
  - Design and test inner loop
  - Design and test outer loop

Nested loops

- print a two-dimensional table:
  - the times table for the numbers 1-10. The outer loop handles each row in turn; the inner loop handles the columns within each row.
- User interface
-
```
   1  2  3  4  5  6  7  8  9  10
1  1  2  3  4  5  6  7  8  9  10
2  2  4  6  8 10 12 14 16 18  20
...
10                            100
```

Nested loops

- Algorithm
  - Print heading
    - While not all columns done
      - Print column number
  - Print table; for each row
    - Print row
      - Print row number
      - For each column
        - » Print column number