Lecture C11: FOR, WHILE and General Loops

Response to 'Muddiest Part of the Lecture Cards'

(45 respondents)

1) General comments on lecture / class. (2 students)
Class is too early:
Sorry not much I can do about that... more than maybe agree that Monday 9am is very early indeed.

Can you post your examples online?
I will put all my in-class-examples in a .zip file and distribute to class.

Where can we find lots of practice problems?
As mentioned in earlier mud response, take a look in the Feldman Ada book or any other Ada book, there are lots and lots of examples to work through, and many of them alos have solutions on the CD or within the book.

2) What is the difference between a Definite and Indefinite while loop? (1 students)
While loops are generally used for indefinite iteration i.e. the number of times the loop has to execute has not been determined. A For loop is used for definite iteration i.e. the number of times the loop has been executed is predetermined. You can replace a For loop with a while loop, but the substitution is prone to error because the tests.

Algorithm to convert a For loop into a While loop:

1. Create a loop variable (in a for loop, the loop variable need not be declared).

2. Initialize the loop variable to the lower_bound of the for loop prior to entering the loop.

3. If lower_bound <= upper bound and 'for I in lower_bound .. upper_bound loop'
    a. The For loop is replaced by 'While (loop_variable <= upper_bound) loop'
    b. Increment loop_variable by 1
    c. Replace every occurrence of the For Loop counter by loop_variable
    d. Rest of program body

4. If (lower_bound > upper_bound) and 'for I in reverse lower_bound .. upper_bound loop'

a. The For loop is replaced by ‘While (loop_variable >= upper_bound) loop’
b. Decrement loop_variable by 1
c. Replace every occurrence of the For Loop counter by loop_variable
d. Rest of program body

```
-- for for_loop_counter in lower_bound .. upper_bound
for I in 1 .. N loop
  --loop body
end loop;

-- replace with
while Loop_Variable <= Upper_Bound loop
  Loop_Variable := Loop_Variable + 1;
  -- loop body
end loop;
```

3) Sentinel and Flag loops? (4 students)

What is their syntax? Are they special kinds of while loops? When is it good to use them? What real-life cases would one use sentinel loops? How is using a flag more effective than other techniques?

Sentinel loops have a special variable called a sentinel. The loop variable is compared to the sentinel to decide the exit condition.

For example:

```
Sentinel : constant Integer := 20;
Number : Integer;

loop
        Ada.Text_IO.Put("Please Enter a number:");
        Ada.Text_Io.Get(Number);

        exit when Number = Sentinel;
        -- do something with the number;
end loop;
```

Here, the loop will exit when the user enters ‘20’. If any other number is entered, the part after the exit condition will be executed. The idea of a sentinel loop is applicable to both general loops and While loops. The same code if written using a while loop will appear as follows:

```
-- variable declarations
Sentinel : constant Integer := 20;
Number : Integer;

-- main program body
      Ada.Text_Io.Put("Please Enter a number:");
```

```
Ada.Text_Io.Get(Number);
while (Number /= Sentinel) loop
            -- do something with the number;
            Ada.Text_Io.Put("Please Enter a number:");
            Ada.Text_Io.Get(Number);
end loop;
```

Note the change in order of the statements in the loop and the change in the exit condition. It still however uses the idea of a sentinel condition.

A sentinel loop is typically used to obtain only a certain set of inputs or to implement a definite loop.

Flag controlled loops on the other hand use a Boolean variable to test the exit condition. The flag variable is set in the loop body when a certain event occurs.

```
--variable declaration
Flag : Boolean;

-- main program body
Flag := False;
while (Flag /= True) loop
        --loop body
            if Event_Occurs then
               Flag := True;
            end if;
          -- more loop body
end loop;
```

Here, the loop will execute until an event occurs. The occurrence of the event sets the Flag to true. For example, another aircraft coming closer than minimum safe separation distance, will cause my program to exit the loop and execute a safety maneuver.

```
-- variable declartion
Aircraft_Too_Close : Boolean;

-- main body

-- do some aircraft processing
         Aircraft_Too_Close := False;

while (Aircraft_Too_Close /= True) loop
            --follow regular flight plan
            if (Other_Aircraft_Comes_Too_Close) then
               Aircraft_Too_Close := True;
```

```
        end if;
      end loop;

if (Aircraft_Too_Close) then
  -- Inform the pilot that aircraft too close
  -- Perform safety maneuver
end if;
```

4) How would incorporate this into a package you are putting together? (1 student)

5) Back to the first slide, what is a "case"? (1 student)
The case statement provides an elegant means of choosing between multiple options (the multiple if statement can be hard to debug).

The general format of the case statement is:

```
case selector is
when value_list_1 =>
    statement(s)_1;
when value_list_2 =>
    statement(s)_2;
...
when others =>
statement(s)_n;
end case;
```

For example:

```
-- variable declaration
type Possible_Choices is (1, 2, 3, 4, 5);
package Choice_Io is new Ada.Text_Io.Enumeration_Io(Possible_Choices);

My_Choice : Possible_Choices;

--main program

Ada.Text_IO.Put("Please Enter Your Choice");
Choice_Io.Get(My_Choice);

case My_Choice is
        when 1 =>
          --execute code for option 1
        when 2 =>
          --execute code for option 2
        when others =>
```

```
        -- execute code for other three options
end case;
```

Here the others option allows the programmer to bundle the code for the options 3,4 and 5 together. The equivalent ‘if-statement’ implementation is shown below:

```
--main program

Ada.Text_IO.Put("Please Enter Your Choice");
Choice_Io.Get(My_Choice);

if My_Choice = 1 then
        --execute code for option 1
elsif My_Choice = 2 then
        --execute code for option 2
else
        --execute code for other three options
end if;
```

In addition to being hard to read, the code above is hard to modify.

6) Comments on examples given on slides. (4 students)

Was the code on the PowerPoint slides in Ada syntax?

No, not on all of them, for example slide 15 "Sentinel-controlled loops" is not Ada syntax, rather "pseudo code".

The program didn't need a capital or lowercase Y or N neccessarily

Not sure what this question really is asking, the program will keep on executing the statements within the loop until a 'y' or 'Y' or 'n' or 'N' is enetered as input to the program.

What was the reasoning why the example of the while loop would not be executed at all?

Are you commenting on slide 15, "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;
```

Here, if the user enters –1, then the loop will not execute because the condition (j/= -1) is not satisfied. The output will be 'Total is –1'

Why is there an empty PUT(" ") in 'times code'?
Only a formating issue. Please test run the code with that line and compare it to when you make that statement into a comment.

7) You said SparkAda requires assert statements - how, and what is assert comments? (1 student)
The state of a program can be captured by the set of values associated with the variables in the program at any point during its execution.

An assert statement checks if the state of the program is consistent with the expected state value.
8) When and when not are Formal paramters needed? (1 student)
Formal parameters are used when subprograms are called to ensure that the right data is passed i. e. if formal parameters are used, then the programmer does not have to ensure that the order in which parameters are passed is correct.
9) General Ada questions? (10 students)
Is there a way to change variable names in a loop, like in: b0, b1 : Integer; then in loop, FOR i in 1..10 loop bi := ... ?
Yes, variables declared and visible within the procedure/function can be changed. You can not on the other hand, change the 'loop variable', in your example: 'i' cannot be changed by the code.

What does (=> width 5) represent?
The minimum amount of spaces that will be used for the printed item.

How do we write/use (for loop_var in low_val .. high_val) loops with incremt steps other than +1
How do you do a step of 2 or more in a loop?
If you want to use a different increment step other than 1, then convert the for loop into a while loop using the algorithm given in question 3. Instead of incrementing by 1, increment by the step you want to use.

Can you use 'i' in operations in a loop? For example:
```
Bob := 2;
for i in 1 .. val loop
bob + 1 = new_number;
```

end loop;
Can you use 'i' in operations in a loop? For example:

```
Bob := 2;
for i in 1 .. val loop
        bob = new_number + i;
end loop;
```

Loop variables can be used on the right-hand side of computations. You cannot change the value of the for-loop variable inside the body of the loop.

In a for statement, does the loop_var need to be defined in the declartions if it is given no value befor the loop?
The loop variable does not have to be defined earlier in the program when using ‘for’ loops.

What is the point of having a package and its body in separate files?
Having the package specification and body in separate files allows multiple programmers to work on the implementation of the same package and have the same view of the interfaces (subprogram calls, data structures).

Can a procedure call itself in Ada?
Yes. A subprogram calling itself is how simple recursion is implemented.
How can I clear the screen in Ada?
Use the Screen package. There is a clear screen function that you can use in that package.

Why do you have to update the loop parameter for while loops?
In a for-loop, the loop variable is incremented automatically. In a while loop however there is no implicit increment operation. It is the responsibility of the programmer to take care of the incrementing.

What happens when I write this:
loop
exit when variable in type;
put(item => "please try again.");
end loop;
the ‘variable in type’ operation checks to see if the variable is of the specified type. The control will pass out of the loop if the variable is in the specified type.

10) "No mud" (21 students)
Well, that is great!