Assignment 6: Loops and conditionals

ETH Zurich

Hand-out: 24 October 2008
Due: 30 October 2008

Goals

- Understand and read loops and conditionals.
- Use loops and conditionals to solve tasks.
- Use nested loops.

1 Reading loops

The structure of a loop contains multiple clauses:

- The **from** clause is required (but may be empty) and specifies the loop initialization instructions.

- The **invariant** clause is optional and contains boolean expressions that are ensured to hold by the initialization instructions and preserved by each execution of the loop clause as long as the exit condition is **False**. This means that the loop invariant also holds when the loop terminates.

- The **variant** clause is optional and contains an integer expression that is setup by the initialization instructions to be a non-negative integer. Every execution of the loop body when the exit condition is not satisfied and the loop invariant is satisfied decreases its
value to a still non-negative value. This means that the loop variant is still non-negative when the loop terminates.

- The `until` clause captures the exit condition of the loop and contains a boolean expression; as soon as the expression returns `True` the execution of the loop is finished. Note that defining the correct exit condition is one of the main challenges in writing loops. Always ensure that the loop does not terminate (1) too early (e.g. forgetting the last iteration), or (2) too late (e.g. executing it once more than intended).

- The `loop` clause contains instructions that are repeatedly executed until the exit condition of the loop is fulfilled. Always ensure that the loop body contains instructions that let the loop advance - forgetting these happens very often and results in endless loops!

The structure of a loop is shown in Listing 1 and an example of a loop in Eiffel is given in Listing 2.

### Listing 1: Loop structure

```plaintext
from
    initialization_instructions

invariant
    invariant_clause

variant
    variant_clause

until
    exit_condition

loop
    loop_instructions

end
```

### Listing 2: Loop example

```plaintext
loop_example is
    -- Execute a loop that prints numbers
    -- from 1 to 100.

local
    count: INTEGER

do
    from
        count := 1
    invariant
        count >= 1
        count <= 101
    variant
        101 - count
    until
        count > 100
    loop
        io.put_integer (count)
        io.put_new_line
        count := count + 1
    end

end
```

**To do**

Assume that the two code extracts in Listing 3 and Listing 4 intend to loop through a list of stations and search for the station named "Cite Universitaire" and highlight it.

1. For each version (Listing 3 and Listing 4) decide whether it does what it is supposed to do.

2. If you think it is not OK, then correct the errors.

You may assume for this exercise that all the entities are not Void (i.e. they are all attached to an object). The feature `start` for container objects sets the internal cursor position to the
beginning of the list, feature item_for_iteration returns the object at the cursor position, feature forth advances the cursor by one position, and after is a boolean query that returns True if the cursor position is past the last element. Note that name = "Cite Universitaire" is not the same as name.is_equal("Cite Universitaire").

To hand in

This is a pen-and-paper exercise: you do not need to write code in EiffelStudio. Hand in your answers and if necessary the corrected versions of Listing 3 and Listing 4.

Listing 3: Version A

```eiffel
explore is
  -- Highlight "Cite Universitaire".
local
  found: BOOLEAN
begin
  Paris.display
  do Paris.stations . start
  until Paris.stations . after or found
  end
  loop
    if (Paris.stations . item_for_iteration .
        name = "Cite Universitaire")
      then
        found := True
      else
        Paris.stations . forth
      end
    end
  end
end
```

Listing 4: Version B

```eiffel
explore is
  -- Highlight "Cite Universitaire".
begin
  Paris.display
  do Paris.stations . start
  until Paris.stations . after or Paris.stations .
    item_for_iteration . name . is_equal ("Cite Universitaire")
  loop
    if (not Paris.stations . after) then
      Paris.stations . item_for_iteration .
        highlight
    end
  end
end
```

2 Equipping Paris

It happens very often that you want to iterate through all the items of a container in Traffic (e.g. through Paris.stations, Paris.lines, or Paris.passengers). To do this you can use the following scheme (here for Paris.lines, similar for the other containers in a TRAFFIC_CITY):

Listing 5: Looping through map item containers

```eiffel
from
  Paris.lines . start
until
  Paris.lines . after
loop
  Paris.lines . item_for_iteration . highlight
  Paris.lines . forth
end
```
To do

1. Download http://se.ethz.ch/teaching/2008-H/eprog-0001/exercises/assignment_6.zip and extract it in traffic/example. You should now have a new directory traffic/example/assignment_6 with assignment_6.ecf directly in it (it is important that the location corresponds to the description here!).

2. Open and compile this new project. Open class LOOPINGS and solve the tasks below.

3. Implement the feature generate_trams_for_line. This feature has a line as argument and should check whether the line is of tram type. To find out whether a TRAFFIC_LINE object is a tram line, you need to create an object of type TRAFFIC_TYPE_TRAM and see if the line’s attribute type is equal to it. If the line is a tram line then the feature should create for every second station a tram that starts moving at this station. So the first tram starts at the first station of the line, the second tram at the third station, the third tram at the fifth station, etc. Use feature set_to_station (a_station: TRAFFIC_STATION) to set the initial position of a tram to a certain station (this feature is available in TRAFFIC_TRAM by inheriting from TRAFFIC_LINE_VEHICLE). Don’t forget to add the generated trams to Paris.

4. Generate trams for all the lines of the city. You may use generate_trams_for_line to achieve this.

5. Implement the feature generate_connecting_bus_line. The idea of this feature is to create a new bus line with \( n \) intermediary stops that connects the given start_station to the end_station. As a first step, you need to create a new line of bus type (use make_with_terminal as creation procedure and the argument start_station as the terminal). Then, in a loop create \( n \) times a new station and extend the line with it. After doing so, add the end_station to the line. The locations of the intermediary stations should be evenly distributed along the straight line between the start_station and the end_station. In the example seen in the figure below, \( n \) was 3, the start station Balard and the end station Mairie d’Issy. To calculate the locations of the newly created stations you need to do some vector calculations based on the locations of the start and end stations. TRAFFIC_POINT provides so called infix-features (+, −, *) that will help you:

<table>
<thead>
<tr>
<th>Vector addition</th>
<th>Vector subtraction</th>
<th>Scalar multiplication</th>
<th>Scalar division</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b, c: TRAFFIC POINT</td>
<td>a, b, c: TRAFFIC POINT</td>
<td>a, b: TRAFFIC POINT f: DOUBLE</td>
<td>a, b: TRAFFIC POINT f: DOUBLE</td>
</tr>
<tr>
<td>c := a + b</td>
<td>c := a - b</td>
<td>b := a * f (Note: the scalar needs to be the second operator)</td>
<td>b := a / f (Note: the scalar needs to be the second operator)</td>
</tr>
</tbody>
</table>

6. Test your implementation of generate_connecting_bus_line with some stations (e.g. Station_balard and Station_mairie_d_issy).

To hand in

Hand in the code of class LOOPINGS.
3 Loop painting

![Image of checkered triangle and diamond]

Figure 1: Example with size 7

To do

1. Write a program that asks the user to input a value and then displays a checkered triangle of the given size as in Figure 1. Be aware that stars and white spaces should be alternating.

2. Extend your application to also display a diamond with same size. Here as well, the stars and white spaces should be alternating.

Hints

You might need to use the integer division operator `//` or the modulo operator `\` for your solution.

To hand in

Hand in your class text.