Assignment 5: Loops

ETH Zurich

Hand-out: 21st November 2005
Due: 29th November 2005

1 Summary

In this assignment you are going to experiment with loops in Eiffel... Please solve this assignment alone.

An example

The syntax of loops is made of several clauses. The from clause is required (but may be empty), it specifies the loop initialization instructions. The variant and invariant clauses are optional. Leaving aside the optional clauses, the execution of a loop consists of executing the initialization_instructions followed by the “loop process”. If the exit_condition is true, the loop process is a null instruction; if it is false, the loop process is the execution of the loop_instructions followed by a new loop process. The structure of a loop is shown in Figure 1.

from
initialization_instructions
invariant
invariant_clause
variant
variant_clause
until
exit_condition
loop
loop_instructions
end

Figure 1: Structure of a loop
Figure 2: Loop example

An example of a loop in Eiffel is given in Listing 2. This code prints 100 numbers on the console (1,...,100).

2 Where is Central?

Goal

- Understand the structure of loops and conditionals.
- Realize the importance of proper stop criteria.

Description

This first part intends to convey the importance of choosing the right stop criteria. In the following class text extract we want to loop through a list of places and search for the place called “Central”. When we have found it, the loop stops and we do something to the place (if it was found). The two code extracts below are supposed to do everything as just described.

To do

1. For each version decide whether it does what it is supposed to.
2. If you think it is not OK, then correct the errors.
Note

1. You may assume for this exercise that all the entities are not Void (i.e. they are all attached to an object).

2. The command set\_found just sets found to either True or False.

```
from places. start
set\_found (False)
until places. after or found
loop
  if (places. item. name = "Central") then
    set\_found (True)
  else
    places. forth
  end
if (not places. after) then
  —— "Perform some operations on the found place"
end
end
```

Figure 3: Version A

```
from places. start
until places. after or places. item. name. is\_equal ("Central")
loop
if (not places. after) then
  —— "Perform some operation on the found place"
end
end
```

Figure 4: Version B

To hand in

This is a pen-and-paper exercise: you do not need to code in EiffelStudio. Hand in your answers and the corrected versions of a) and b) (if necessary).
Solution

Concerning Version A:

- First, the equality operator will always return \textit{False}. \textit{STRING} objects are not expanded, and = tests for reference equality, not value equality. We need to use \textit{equal} or \textit{is_equal}.

- Second, the if-statement is inside loop: the loop will perform some operations on every object until the loop stops.

- The corrected code of version A is shown in Listing 5.

Concerning Version B:

- Endless loop: there is no call to a command that advances the cursor position in the list.

- Possible precondition violation: \textit{places.item.name.is_equal("Central")} may be tested before \textit{places.after}. In the case where \textit{places.after} holds, the call to \textit{places.item} may violate the precondition \textit{not off}: \textit{not off} of feature \textit{item} in class \textit{LINKED_LIST}.

- The corrected code of version B is shown in Listing 6.
from
    places.start
set.found (False)
until
    places.after or found
loop
    if (places.item.name.is_equal ("Central")) then
        set.found (True)
    else
        places.forth
    end
end
if (not places.after) then
    -- "Perform some operations on the found place"
end

Figure 5: Corrected Version A

from
    places.start
until
    places.after or else places.item.name.is_equal ("Central")
loop
    places.forth
end
if (not places.after) then
    -- "Perform some operation on the found place"
end

Figure 6: Corrected Version B
3 Fancy graphics

Goal

- Play around with loops and conditionals.
- Be creative and make FLAT_HUNT look nicer.

Description

In class PLAYER_DISPLAYER in FLAT_HUNT, there is a feature called mark_defeat. This feature is called on the estate agent when the hunters find him, or is called on all the hunters if the agent can escape. Up to now, mark_defeat just draws a black circle. There is a loop prepared, but for now, this loop is empty. Your task is to fill this loop: try to make some nicer graphics whenever the game is over. This could, for example, look like Figure 7. However, instead of circles, you might also want to use lines or rectangles. Another idea is to play with colors... It would probably look even better if you would add some conditionals.

![Figure 7: Better graphics...](image)

Note

- Make sure that all assertions checking in Project -> Configuration are enabled.
• Have a look at the comments directly in the source code.

• Write the variant and invariant clauses.

To hand in

• Hand in your version of feature mark_defeat, plus a screenshot of your animation in a reasonable graphical format (i.e. jpg, png, gif, etc.). Please, no bitmap (bmp).

• Don’t forget to upload your learning logs.

Solution

Download the source code:

http://se.inf.ethz.ch/teaching/ws2005/0001/exercises/player_displayer.e
4 Loop painting

Goal

- To try nested loops.

Description

You can use loops within loops to display certain figures, like the one in Figure 8.

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 8.
2. Be aware that stars and white space should be alternating.

To hand in

Hand in your class text. Don’t forget to upload your learning logs.

Solution

Download the source code:

http://se.inf.ethz.ch/teaching/ws2005/0001/exercises/checker_triangle.e