Assignment 6: Loops and conditionals

ETH Zurich

Hand-out: 22 October 2010
Due: 2 November 2010

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER “NOT COMPUTER PEOPLE.”

WE DON’T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE’RE USUALLY JUST DOING THIS:

PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN. CONGRATULATIONS; YOU’RE NOW THE LOCAL COMPUTER EXPERT!

Tech Support Cheat Sheet © Randall Munroe (xkcd.com)

Goals

- Practice loops and conditional instructions.
1 Reading loops

To review the structure and semantics of loops refer to section 7.5 of Touch of class.

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 1: Looping through map item containers

from Paris.lines.start -- Move cursor to the first element;
until Paris.lines.after -- until cursor is beyond the last element
loop Paris.lines.item_for_iteration.highlight -- do something with the element at the current
cursor position
Paris.lines.forth -- and advance the cursor.
end

To do

Assume that the two code extracts in Listing 2 and Listing 3 intend to loop through a list of stations and search for the station named “Cite Universitaire” and highlight it.

1. For each version (Listings 2 and 3) decide whether it does what it is supposed to do.

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

Hints

Operator = used on expressions of a reference type compares two references. If they are pointing to the same object the result is true, otherwise is false. In contrast, feature is_equal compares the content of two objects.

For this exercise you may assume the following:

- There are no compilation problems
- All the entities are not Void (i.e. they are all attached to an object)

To hand in

This is a pen-and-paper exercise: you do not need to write code in EiffelStudio. Hand in the corrected versions of Listing 2 and Listing 3.

2 Equipping Paris

In this task you will be exercising loops in the context of Traffic.

To do

1. Download http://se.ethz.ch/teaching/2010-H/eprog-0001/assignments/06/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.
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 an argument. If the line is of tram type then the feature should create for every second station a tram that starts moving at this station. So the first tram should start 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 to set the initial position of a tram. Don’t forget to add the generated trams to Paris and don’t forget to add contracts.

4. Implement feature equip to generate trams for all the lines of the city. To achieve this use generate_trams_for_line.

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. Create a new line starting at start_station, then use a loop to create \( n \) new stations and extend the line with them, and finally 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, \( n \) is 3, the start station is Balard and the end station is 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 some so called infix-features \(+, -, \times\) that will help you:
Vector addition | a, b, c: TRAFFIC_POINT | c := a + b
Vector subtraction | a, b, c: TRAFFIC_POINT | c := a - b
Scalar multiplication | a, b: TRAFFIC_POINT, f: DOUBLE | b := a * f (Note: the scalar needs to be the second operator)
Scalar division | a, b: TRAFFIC_POINT, f: DOUBLE | b := a / f (Note: the scalar needs to be the second operator)

Another hint is that you may want to perform some rounding on the computed locations (see feature rounded in class DOUBLE). Don't forget to add the line and the stations to Paris and don't forget to add contracts.

6. Test your implementation of generate_connecting_bus_line with some stations (e.g., Station_baland and Station_mairie_dissy), adding code to feature equip.

To hand in
Hand in the code of class LOOPINGS.

3 Loop painting

To do
Write a program that does the following:

1. Asks the user to input a positive integer.

2. Displays, using asterisks, a checkered rectangle triangle having as hypotenuse a number of asterisks equal to the user input (see Figure 1.). Stars and white spaces should be alternating.

3. Displays, using asterisks, a diamond having as side the same number of asterisks as the user input. Here as well, stars and white spaces should be alternating.

4. Take into consideration that the user might not always input values you expect. Make sure your program does not crash, no matter what the user inputs.

To hand in
Hand in your class text.

4 Programming a boardgame: Part 2

In this task you will implement a given set of classes. They may not coincide with the ones you picked last week, but it is easier to go on altogether in this way.

As a reminder, you will find below the description of the problem. It has been slightly modified because it mentions six-sided dice. While this is a little detail, it gives you an idea of
the fact that across different iterations of the design and development process the specifications can actually change.

The board game comes with a board, divided into 40 squares, a pair of six-sided dice, and can accommodate 2 to 6 players. It works as follows:

- All players start from the first square.
- One at the time, players take a turn: roll the dice and advance their respective tokens on the board.
- A round consists of all players taking their turns once.
- The winner will be the player that first advances beyond the 40th square.

**To do**

Implement the prototype of the boardgame using the following classes:

- *GAME*: encapsulates the logic of the game (start state, the structure of a round, ending conditions).
- *DIE*: provides random numbers in the required range.
- *PLAYER*: stores the state of each player in the game and performs a turn.

You can also use class *APPLICATION* as root class of your system, which is responsible for interaction with the user.
To hand in

Submit the code of classes \textit{GAME, DIE, PLAYER, APPLICATION}.

If you feel lost...

If you tried really hard but you don’t have a clue on how to organize the given classes internally, you may want to download the class skeletons (with empty feature bodies) from http://se.ethz.ch/teaching/2010-H/eprog-0001/assignments/06/board_game.zip