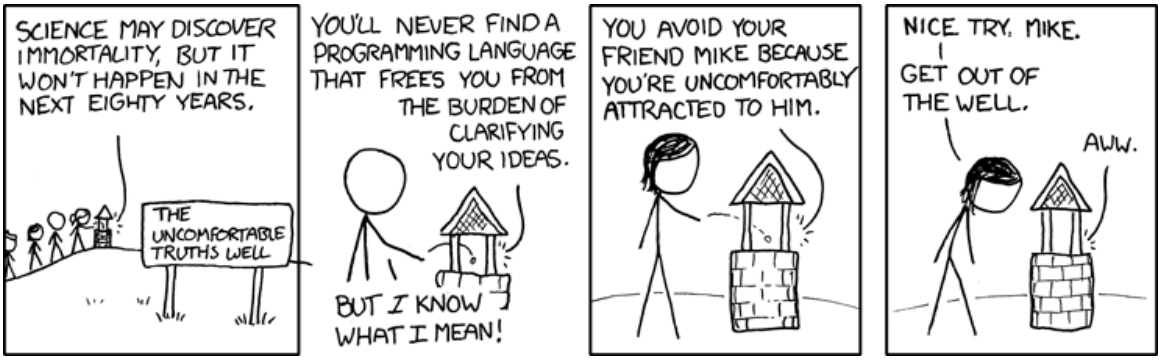
Assignment 7: More Peachy Loops

ETH Zurich

Hand-out: 5 November 2010
Due: 16 November 2010



Well 2 © Randall Munroe (xkcd.com)

Goals

- Practice loops and conditionals.
- Learn to deal with files for input and output.
- Understand the importance of unit testing your application.

1 Buildings for Paris

In this task, you will generate buildings for Paris that are placed along the roads. Class *TRAFFIC_ROAD* represents streets, rail tracks, or light-rail tracks. A *TRAFFIC_ROAD* can be a one-way road in which case it only contains one *TRAFFIC_ROAD_SEGMENT* (accessible through the feature *one_way*) or a two-way road in which case there are two segments (feature *one_way* and feature *other_way*).

A *TRAFFIC_ROAD_SEGMENT* contains a set of points defining the shape of the segment. The points are accessible through the feature *polypoints*. Figure 1 illustrates this. To iterate through the polypoints of a road segment, you can use the following scheme:

```
local
  road: TRAFFIC_ROAD
  segment: TRAFFIC_ROAD_SEGMENT
  point: TRAFFIC_POINT
  i: INTEGER
```

```
do
  road := Paris.roads.item (1) -- Access the first road in Paris
  segment := road.one_way -- Access the road segment
  from
    i := 1
  until
    i > segment.polypoints.count
  loop
    point := segment.polypoints.item (i)
    -- Do something with 'point'
    i := i + 1
  end
end
```

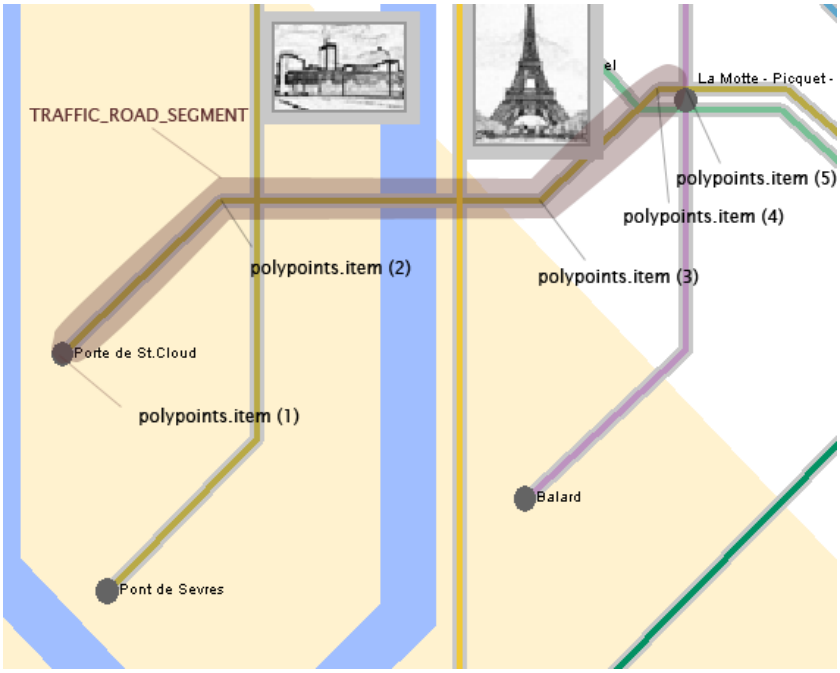


Figure 1: Example of a *TRAFFIC_ROAD_SEGMENT* and its *polypoints*.

To do

1. Download http://se.ethz.ch/teaching/2010-H/eprog-0001/assignments/07/assignment_7.zip and extract it in `traffic/example`. You should now have a new directory `traffic/example/assignment_7` with assignment_7.ecf directly in it.

2. Open and compile this project. Open class *CONSTRUCTION*.

3. Implement feature *generate_buildings_along_segment*. It should generate buildings of type *TRAFFIC_VILLA* along the road segment given as argument. Width, depth and height of each building should be equal to 16.0. Buildings should be located on both sides of the road. The distance between the building center and the road, as well as between the centers of two adjacent buildings, should be equal to 24.0. Fit as many buildings between

each pair of points as you can for the given distance (the centers of all buildings should lie between the points). Figure 2 shows a schematic image of it.

Test your implementation of *generate_buildings_along_segment* by calling it from feature *build* with any road segment of your choice. Don't forget to add the generated buildings to *Paris*.

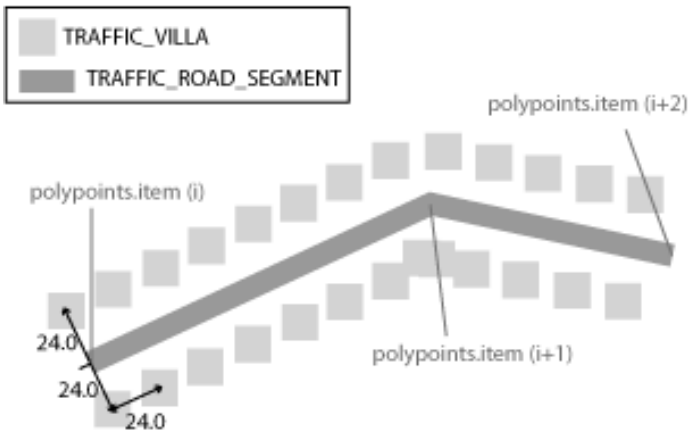


Figure 2: Buildings along a road segment

4. Implement feature *generate_buildings* that generates buildings along all roads in Paris. Call this feature from feature *build*.

5. **Optional:** The algorithm you implemented in Step 4 will put buildings on top of existing buildings and public transportation lines. To fix this you can use *TRAFFIC_GRID*. The grid contains $n \times n$ fields that can be marked as either free or occupied. The feature *fill_grid* of class *CONSTRUCTION* creates a grid and marks the locations of public transportation lines as occupied. Before generating a building you can check whether its location is already occupied using feature *has_rectangle_collision*.

To hand in

Hand in the code of class *CONSTRUCTION*.

2 Bagels!

To do

You are to write a program that plays a game called Bagels. It is a variation of Mastermind that uses the digits 1 to 9 (no zeros). Here is the specification:

- The player specifies a positive number n. The program generates an n-digit number that the player will try to guess.
- The program asks the player to provide a guess. Only n-digit numbers with no zero digits are accepted.
- After each guess, the program gives clues as to how close the player is to getting the right answer.

  - For each right digit in the right position, it will say `Fermi`.
  - For each right digit in the wrong position, it will say `Pico`.
  - If there are no right digits, it will say `Bagels`.

- The program should list all of the `Fermi`s first and then all of the `Pico`s. Because of this, the player can't assume anything from the order of the clues.

Have a look at the following examples:

Answer	Guess	Report	Comment
123	329	`Fermi Pico`	-
123	312	`Pico Pico Pico`	-
999	912	`Fermi`	Each digit matches only once
912	999	`Fermi`	-
222	262	`Fermi Fermi`	-
112211	191191	`Fermi Fermi Pico Pico`	Two 1s are `Fermi`s and 2 are `Pico`s

Hints

It's a good idea to use a string (instead of an integer) to store the answer and the guess.
To generate a random answer, use the class *RANDOM*.

To hand in

Hand in the code of your class.

3 Decimal to binary converter

For this task you will be using peach[3], a web-based assignment submission platform[1].

To do

1. Register on http://ext.peach3.nl using the link on the top left. As login name, please use your first and last name.

2. Once you are successfully logged in, click on the "course manager" link on your home page and then click on the "Join a course" button on the "My Courses" page. Go to the folder ETH Zürich/252-0001-00 Einführung in die Programmierung (2010-2011) and select your exercise group from the list.

3. On the left of the screen, you should have a "Courses" window with course you just subscribed to. Click on the course. On the right a new course window will appear. Under "Available Assignments" you should see "Decimal-to-binary". To read the assignment text click on "Description".

4. Create a project in EiffelStudio and solve the exercise. To test your program locally you will need to manually create a text file "converter.in", containing the input to the program, in the same directory as your ".ecf" and ".e" files (when testing on peach[3] the input file will be provided by the server automatically). Make sure that the format of your output file is exactly as in the problem description, otherwise it won't be accepted by the system. Note that it is an error to create an output file if the input was incorrect.

[1] Erik Scheffers, Tom Verhoeff, Stefan Geuns and Marijn Kruisselbrink.

5. Go back to the "Assignments" window in peach[3], select "Decimal-to-binary" and click on "New submission". Upload two files: "decimal_to_binary_converter.e" and the ".ecf" file; click "Save".

6. You can check the status of your submission in the "Submissions" window. If it is not there, try clicking "Refresh" (upper-right corner). Double-clicking on the submission shows detailed information about individual tests.

7. You can resubmit as many times as you want. We encourage you to submit until you pass all the tests.

Hints

Use the class *PLAIN_TEXT_FILE* to work with files. In the simplest case (without error handling) you can use features *make_open_read*, *read_line* and *close* for reading from the input file and features *make_open_write*, *put_string* and *close* for writing to the output file. Don't forget to close all the files you open.

To hand in

Hand in the code of your class *DECIMAL_TO_BINARY_CONVERTER* and a screenshot showing the number of tests passed.