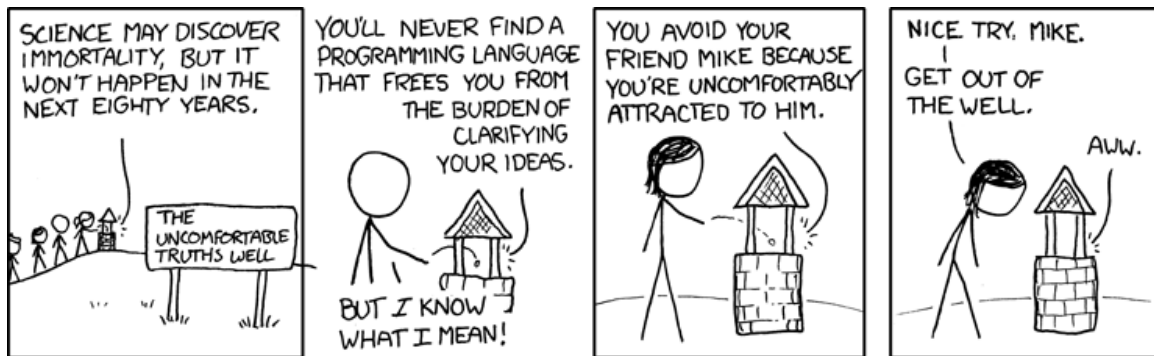


## Assignment 7: More Peachy Loops

ETH Zurich

Hand-out: 30 October 2009

Due: 10 November 2009



Copyright Randall Munroe <http://xkcd.com>

### Goals

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

### 1 Buildings for Paris

In this task, you will generate buildings for Paris that are placed along its roads. *TRAFFIC-ROAD* represents streets, railtracks, or lightrail tracks that public transportation lines use as their medium to operate on. 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 *TRAFFIC-ROAD-SEGMENT*s (feature *one\_way* and feature *other\_way*). A *TRAFFIC-ROAD-SEGMENT* connects an *origin* (starting station) with a *destination* (end station) and contains a set of points defining the shape of the segment. All two-way roads of the Paris map have the same set of points for the first and the second segment (but in inverted order). The points are accessible through the feature *polypoints* in *TRAFFIC-ROAD-SEGMENT* (inherited from *TRAFFIC-SEGMENT*). Figure 1 illustrates this. To iterate through the polypoints of a road segment, you can use the following scheme:

local

```
road: TRAFFIC-ROAD
seg: TRAFFIC-ROAD-SEGMENT
coord: TRAFFIC-POINT
```

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

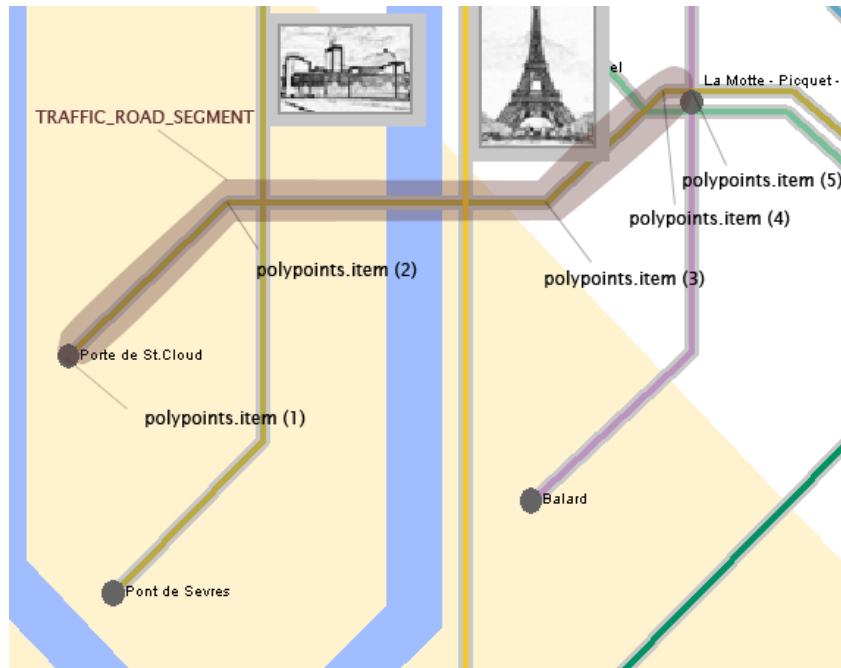


Figure 1: Example of a *TRAFFIC\_SEGMENT* and its *polypoints*.

## To do

1. Download [http://se.ethz.ch/teaching/2009-H/eprog-0001/exercises/assignment\\_7.zip](http://se.ethz.ch/teaching/2009-H/eprog-0001/exercises/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 (it is important that the location corresponds to the description here!).
2. Open and compile this new 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. The result should be two polylines of buildings parallel to the polyline of the road segment: each of them should

lie on opposite sides of the road segment, at distance 24.0. The distance between two buildings next to each other on a row, to be interpreted as the distance between their centers, should also be 24.0. Figure 2 shows a schematic image of it. Note that each consecutive point pair *polypoints.item (i)* and *polypoints.item (i+1)* defines a straight line. The first two buildings for each point pair are placed on the perpendicular axis to this straight line going through *polypoint.item (i)*. The number of buildings for each point pair and row is the rounded down number of buildings that would fit on the line defined by the point pair. Test your implementation of *generate\_buildings\_along\_segment* by calling it from feature *build* with an object of type *TRAFFIC\_ROAD\_SEGMENT* as argument (see the code above for hints on how to access a road segment). Don't forget to add the generated buildings to *Paris*.

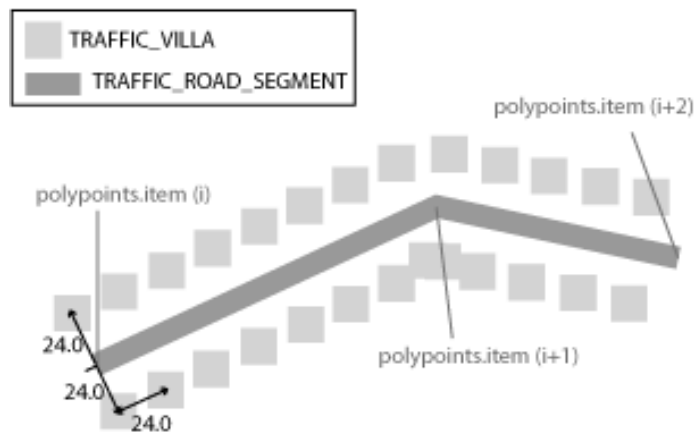


Figure 2: Buildings along a road segment

4. Implement feature *generate\_buildings* that does the same for all roads in Paris. Call *generate\_buildings* in feature *build*.
5. **Optional:** The algorithm you implemented in Step 4 will put buildings on top of existing buildings and public transportation lines. To prevent this you can use a *TRAFFIC\_GRID*. The grid needs to be the same size as Paris (using its center and radius) and contains  $n * n$  fields that can be marked as occupied. The feature *fill\_grid* of class *CONSTRUCTION* creates and initializes such a grid for you and you can use it to create and add a building to Paris only if it doesn't collide with an existing item. See the comment at the end of feature *generate\_buildings\_along\_lines* for an explanation on how to do this.

## Hints

Again, you will need to do vector calculations that are available from class *TRAFFIC\_POINT*. For a short description of this class, have a look at assignment 6. Additionally, you might find the feature *normalize* of *TRAFFIC\_POINT* useful to get a vector of length 1.

## To hand in

Hand in the code of class *CONSTRUCTION*.

## 2 Bagels!

### To do

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

- The program generates an n-digit number that the user will try to guess. The user is asked at game startup with what value of "n" he or she wishes to work.
- The program asks the user to provide a guess. Only n-digit, non-zero numbers are accepted for every guess.
- After each guess, the program gives clues as to how close the user is to getting the answer right. If there is a right digit in the right position, it will say FERMI. If there is a right digit but in the wrong place, it will say PICA. If none of the two situations above is verified, it will say : "No FERMI and no PICA! ".
- The program should list all of the FERMIs first and then all of the PICAs. Because of this, the user can't assume anything from the order of the clues.

Have a look at the following examples:

Answer	Guess	Report	Comment
123	329	FERMI PICA	-
123	312	PICA PICA PICA	-
999	912	FERMI	Each digit matches only once
912	999	FERMI	-
222	262	FERMI FERMI	-
112211	191191	FERMI FERMI PICA PICA	Two 1s are FERMIs and 2 are PICAs

### Hints

The class *STRING* provides many useful features for string manipulation and we encourage you to store the number to guess as an object of type *STRING* instead of as an *INTEGER*. To generate a random answer, you will have to use the class *RANDOM* again.

### To hand in

Hand in your class text.

### 3 Decimal to Binary conversions

For this task you will be using peach<sup>3</sup>, a web platform meant, among other things, to manage submissions of programming assignments<sup>1</sup>.

#### To do

1. Connect to <http://ethz.peach3.nl/>. Register using the appropriate 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 at the center of 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 (2009-2010) and select your exercise group from the list.
3. On the left of the screen, you should have a "Courses" window, and under it you should now see the 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 converter". The text of the assignment is visible by clicking the button labeled "Description", at the right of the "New submission" button.
4. Create a project in EiffelStudio and write the solution to the exercise. You will need to create your own input file called "converter.in" to be able to test it locally. Compile and test it until you are happy with it. Before submitting, remember to comment the lines that create the input file. You won't be needing them in the submitted solution because the input files are already on the server, ready to challenge your solution!
5. Go back to peach, to the window in which you can see the line with the "Decimal-to-binary converter" assignment. By clicking on the assignment, the button "New submission" will become active. By clicking on it you will be able to submit your solution to the assignment, in the form of a decimal\_to\_binary\_converter.e file and the project (.ecf) file. After clicking on the "New submission" button a new window will open. Click "add" to submit the first file, then again to submit the second (they will appear under "Files") and finally press "Save". Don't submit any other file! The input test files are already on the server, and also the executables, because your source file will be compiled and executed directly there, so no EIFGENs directories have to be submitted.
6. You can check the status of your submission in your home page (tab "home"). By clicking on the big area at the bottom of the home page you will get back to your course page and get information about your submission (tests passed and failed). If your submission is still not there, try clicking the refresh button on the top right of the submission window. Select the submission you are interested in and click on the bottom button in the shape of an eye to have a view of all the tests. Clicking on the single tests you can have more detailed information, like the state of the test (Passed/Failed), the score, and a report including the contents of the input and output files.
7. You can resubmit as many times as you want. The points that the system shows you for each test (either 0 or 10) are meant for feedback, not for grading. We therefore encourage you to submit until you pass all the tests!

---

<sup>1</sup>Erik Scheffers, Tom Verhoeff, Stefan Geuns and Marijn Kruisselbrink contributed to peach<sup>3</sup>, which is used by Eindhoven University of Technology, The Netherlands, and for the Nederlandse Informatica Olympiade.

## Hints

Objects of class *PLAIN\_TEXT\_FILE* can be used to model files. There you can find features to perform the fundamental operations on files, such as

- Open (for writing or reading)
- Read
- Write
- Close

## To hand in

Hand in the source code of your class *DECIMAL\_TO\_BINARY\_CONVERTER* and a screenshot showing the number of tests passed.