UNIVERSITÀ DI PISA

Università degli Studi di Pisa

Laurea Magistrale in Informatica

Advanced Programming

Final Term Paper

Start Date: 11/01/2018
Submission deadline: 31/01/2018 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone's code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C# and Java.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary. **Do not include in an exercise code only needed for a later exercise**.
4. cite references to literature or Web pages from where information was taken.

Introduction

In this project, you will develop a system for describing the layout of the elements in graphical interface using constraints expressed in a DSL language, called VFL. The elements of the graphical interface are called *views* and can be nested. The language allows specifying spacing and dimensions, relative positioning, either vertical or horizontal. Here is an example of a layout constraint program:

```
view panel
V:[menuBar]-[textArea]
H:|-30-[index(100)][textArea]-30-|
.

view menuBar
[file][help]
.
```

The first view is called `panel`, and includes three views, called `menuBar`, `textArea` and `index`. The `menuBar` view consists of two other views: `file` and `help`. Views can be connected with other views at a specified spacing with the notation `-30-`, indicating a distance of `30` pixels, or to a fixed spacing by a single dash (`-`), or with no spacing in the absence thereof. The vertical bar (`|`) indicates the border of the containing view. The notation `H:` or `V:` in front of a constraint indicates a horizontal or vertical layout, respectively. Properties of a view, for example width in a horizontal constraint, are specified in parenthesis: `index(100)` means that the width of the index should be 100 pixels.

Exercise 1

Design a set of classes to represent expressions in the constraint language.

Exercise 2

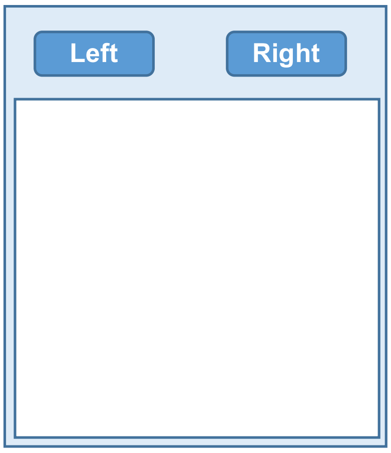
Implement a recursive descent parser for VFL, without using a parser generator. Ensure that each parser method returns a parse tree for the input it analyzes.

Exercise 3

Write a layout generator, which, given a VSL specification and the width and height of the containing view, computes positions and dimensions for all views satisfying the constraints. The sizes of views should be as large as possible within those constraints. For example, since `index` and `textArea` are anchored to the borders of the containing view, the width assigned to the `textArea` should be enough to fill all the horizontal space.

Exercise 4

Write the layout for the following example, where the buttons should stay at the top, evenly spaced with the neighboring elements:



Show the dimensions and positions of each element produce by the layout generator expressed in pixels as:

```
elementName: x, y, width, height
```

relative to the coordinate of the containing view. Show the results for the layout within two views of different horizontal and vertical size.

Exercise 5

Extend the language and generator, to allow expressing inequality constraints, for example:

```
|-30-[index(100)]-[textArea(>200)]-30-|
```

The `textArea` can have any size `> 200` pixels. If there are more than one such inequality in a constraint, the space left after fulfilling the requirements should be equally divided among the elements.

Exercise 6

Describe the delegate event model used in a typical graphic framework. Illustrate with a simple example the benefits of using abstractions like delegates or inner classes in such frameworks. Compare the scoping rule limitations of inner classes with respect to traditional function closures.