Università degli Studi di Pisa

Laurea Magistrale in Informatica

Advanced Programming

Final Term Paper

Start Date: 02/06/2015
Submission deadline: 24/06/2015 (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

We will develop a DSL for generating interactive wizards. For example the following wizard allows configuring a mail client:

```
wizard MailSettings {
  {
    client: "Client:" ["Thunderbird", "Outlook", "Apple Mail"],
    type: "Type:" ["POP", "IMAP"]
  },
  {
    port: "Port:" if (type == "POP") {
          [25, 125]
          } else {
```

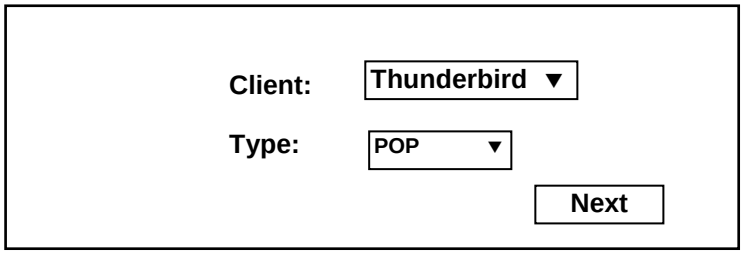
```
            [993, 995]
            }
        Security: "Security:" ["None", "SSL"]
      }
    }.
    {
      set_client: "Client:" client,
      set_type: "Type:" type,
      set_port: "Port:" port,
      set_security: "Security:" security
    }
  }
```

Each element represent a step for setting some variables. Each variable consists of a variable name, label and type. The type can be an enumeration represented by an array of values or an expression for computing the variable value.

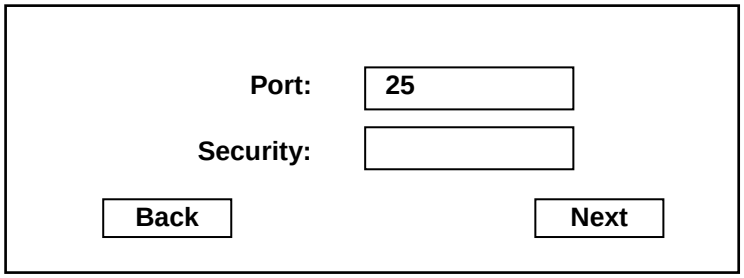
Expressions should include booleans (e.g., &&, || and !), comparisons (<, > and ==) and arithmetic (+, -, *, / and ^). The allowed types are: boolean, string, integer, real and enumeration. Notice that the result of an expression can be a type.

This form definition should generate a GUI allowing the following interaction:

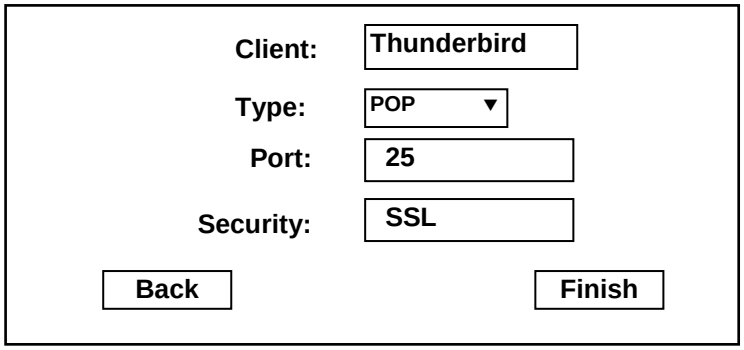
Step 1



Step 2



Step 3



Exercise 1

Design a hierarchy of classes to represent DSL wizards.

Exercise 2

Write a recursive descent parser for analyzing the DSL.

Exercise 3

Write a set of widgets that implement the GUI, for example using JavaScript and jQuery.

Exercise 4

Write a compiler that generates code for a form using the widgets. Show the generated code for the example.

Exercise 5

Extend the solution to handle numerical expression, for example to select a flight from a place to a destination, display alternative flights with different times and prices, choosing one, setting the number of passenger and displaying the total amount. with associated price, to be shown in the price field. Computed values could be specified as follows:

```
price: "Price:" real,
passengers: "Passengers:" integer,
amount: "Amount:" real(price * passengers)
```

Updates to computed values should occur as soon as new values are entered in the fields of the form.
Show a full example of usage.

Exercise 6

Consider defining classes for representing matrices and vectors of floats. If the language does not provide overloading of operators, operations on these classes must be defined as methods. For example method `Matrix sum(Matrix m)` for adding two matrices and method `Vector dot(Vector v)` for multiplying a matrix by a vector. Multiplying the sum of two matrices `A` and `B` by `Vector v`, would be expressed as:

```
Vector r = A.sum(B).dot(v)
```

Discuss how using template metaprogramming an expression similar to the above could be transformed into code that will avoid both allocating the intermediate matrix sum and using two loops to evaluate the expression. Is there any other way to achieve a similar code optimization?