*Università degli Studi di Pisa*

**Laurea Specialistica in Informatica**

# Advanced Programming

## Final Term Paper

**Start Date: 28/06/2017**
**Submission deadline: 20/07/2017 (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 other students 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.
4. cite references to literature or Web pages from where information was taken.

## Introduction

In this project, you will develop a Cloud Deployment Engine (CDE), used to deploy applications to a cloud provider. The provider can be asked to provision a VM (called an agent below) of a given kind, or to terminate a VM. The engine reads a declarative specification of the services to be deployed, that includes relations among the agents. The description is expressed in a Yaml file like this:

```
services:
  moodle:
    units: 1
  database:
    units: 1
agents:
  moodle:
    class: Moodle
```

```
      events:
        init:
          handler: install
        connect:
          after: [init, db_available]
          handler: connect
        start:
          after: [connect]
          handler: start
      requires: [db]
    database:
      class: Mysql
      events:
        init:
           handler: install
        start:
          after: [init]
          handler: start
      provides: [db]
  relations:
    - [moodle: db_available, database: init]
```

The application consists of two services (`moodle` and `database`), each one in a single instance (`units`). The two services are described in the `agents` section. It specifies the class that implements the service and the events that may occurr during the lifetime of the service.

Events are triggered when all the precondition events in the `after` clause have been handled, and cause the action named in the handler parameter. Actions are performed asynchronously and the agent communicates back to the CDE when it is completed. For example, the `connect` event, specifies the handler `connect`, which must be triggered when both the `init` and `db_available` events have been handled. The `db_available` event is triggered explicitly by the CDE to establish the relation between two agents, listed in the relations section.

`[moodle: db_available, database: init]` specifies that the `db_available` condition can be met for the `moodle` agent after the `init` event has been handled by the `database`.

The CDE keeps information about the state of services. When given a description of the services to deploy, the CDE compares the required state with the current one, and determines the actions to perform to achieve that state. For example, initially there will be no services running, so it will determine that it needs to create one instance each of the agents `moodle` and `database`. It then determines which events can be triggered and invokes their handlers.

The cloud provider receives requests for instantiating an agent of a given kind and creates instances that communicate back to the CDE, notifying when an event has been handled. Each agent should run in its own thread.

## *Exercise 1*

Design a set of classes to represent the syntactic constructs of CDE Yaml definitions (e.g. Services, Relations, Agents, etc.).

## *Exercise 2*

Implement a recursive descent parser for CDE without using external libraries or parser generators. Split the parser into a lexical analyzer and a syntax analyzer, as presented in the slides of the course.

## *Exercise 3*

Implement the cloud provider.

## *Exercise 4*

Implement the CDE as an event driven applications that accepts deployment requests and notifications from agents. Use a generator for producing the list of events that must be triggered at each time.

Perform a simulation of the example in the introduction, assuming that the handlers have a random time duration and print the list the events triggered in a sample run.

## *Exercise 5*

Extend the CDE so that if given a new description, will perform the necessary actions to reach the required state. For example, if the number of units is increased, it will create the additional units and create the additional relations among the new agents. If units decrease, the extra ones should be terminated.

## *Exercise 6*

Explain the notion of lexical closure and list some programming languages that support them. Explain the relations between lexical closures and C# delegates and methods in Java inner classes. Provide an example of a realistic use of a closure in JavaScript in an AJAX application.

.