Università degli Studi di Pisa

Laurea Magistrale in Informatica

Laurea Magistrale in Informatica e Networking

Advanced Programming

Final Term Paper

Start Date: 20/06/2012
Submission deadline: 21/07/2012 (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 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.
4. cite references to literature or Web pages from where information was taken.

Introduction

Consider a notation for expressing the call graph of several functions, like the following:

```
x = foo(a, g(y));
y = h(foo(b));
z = f(a, x);
```

Esercise 1

Define some classes for representing a call graph, avoiding using links in both directions.

Esercise 2

Provide an LL(1) grammar for the language expressing the call graph and implement a recursive descent parser for the language.

Esercise 3

Assuming that there are no cycles in the call graph, implement an enumerator that lists one at a time all possible topological orderings among the calls that satisfy the ordering of dependencies.

Esercise 4

Extend the previous enumerator to perform a check on the presence of cycles.

Esercise 5

Extend the classes of Exercise 1, in order to represent the duration the invocation of each function. Provide a method, which given a number *n* of processors, computes the sequence of calls to assign to each processor so that the overall execution time is minimized.

Esercise 6

Extend the previous classes by adding the information on whether a call has been performed and provide an enumerator for those calls that still need to be performed. Provide a solution that avoids rewriting the whole enumerator.

Esercise 7

Explain the notion of lexical closure and list some programming languages that support them. Show how partial applications can be obtained if lexical closures are available and viceversa. Explain the relations between lexical closures and C# delegates and methods in Java inner classes. Provide an example of use of a closure in JavaScript in an AJAX application.