FOUNDATIONS of COMPUTER PROGRAMMING

(teachers/lecturers: Claudio Mirolo and Dario Della Monica)

Course Aims

- Introducing the main scientifical and methodological foundations of computer programming.
- Understanding the nature, potentials and limits of programming.
- Developing basic problem solving and operational skills for programming in-the-small.
- Learning how to apply basics concepts in the functional, imperative and object-oriented paradigms in order to solve simple problems.
- Understanding the algorithmic logic ensuring the correct behaviour of a program.
- Understanding basic concepts of data abstraction and object-oriented programming.

At the end of the course the student should have acquired the basic knowledge and skills to design, organize and write small-scale programs, developed according to the functional, imperative and object-oriented paradigms; moreover, he/she should be able to analyze the algorithmic logic to verify a program's correctness with respect to formal specifications.

Course Organization

The course focus is on the main forms of abstraction to cope with problem complexity: procedural abstraction, data abstraction, and abstraction of state. The procedural and data abstraction are introduced through suitable examples from a functional perspective; the abstraction of state from an imperative and object-oriented perspective. The laboratory sessions are aimed at developing and experimenting with programs that apply the techniques presented in the theoretical lessons.
The course is based on a *functional-first* approach (IEEE-CS/ACM Computing Curricula 2001).

Prerequisites:

High school mathematics.

Course Syllabus

Part I - Procedural abstraction (*Scheme*)

Numerical and non-numerical expressions. Functional approach: procedural abstraction. Simple- (*if*) and multiple-choice (*cond*) constructs. Numerical, boolean, character and string values. Recursive procedures. Well-founded recursive definitions. Evaluation model via substitution and reduction. *Let* construct. General and tail recursion. Tree recursion and computational complexity. Correctness of recursive programs: proof by induction. Higher order procedures.

Part II - Data abstraction (*Java*)

Simple data abstractions. Basic data structures: pairs, lists, and procedures on lists. Different implementations of abstract data types. Data structures from the user's (protocol/interface) vs. the implementor's viewpoint (behavior).

Part III - Abstraction of state (*Java*)

Concept of state and imperative paradigm. Basic statements and constructs of the *Java* language. Arrays and array operations. Data structures and imperative approach. Top-down (memoization) and bottom-up dynamic programming. Functional vs. imperative approach. State abstraction: concepts of class, object, constructor and method. Object interface, encapsulation and information-hiding in the object-oriented approach. Examples of implementation of dynamic data structures in *Java*. Verification of correctness: Assertions, loop invariants and termination functions.

Recurrent Concepts: Procedural abstraction; computational model; recursive approach; tail recursion; tree recursion; imperative approach; iteration; algorithm; computational complexity; preconditions and postconditions; invariant; termination; data abstraction; state abstraction; object-oriented approach; protocol; object; data hiding; modular organization.

FOUNDATIONS of COMPUTER PROGRAMMING

(teachers/lecturers: Claudio Mirolo and Dario Della Monica)

Laboratory

The laboratory sessions are about design, development and experimentation of small-scale programs; they are meant to stimulate students' organization skills as well as their ability to work autonomously.

Textbooks

- Max Hailperin, Barbara Kaiser, Karl Knight
  Concrete Abstractions: An Introduction to Computer Science Using Scheme
  Brooks/Cole Publishing Company, 1999 (ISBN: 0-534-95211-9)
  https://books.google.it/books?id=yYyVRueWlZ8C
- Robert Sedgewick, Kevin Wayne
  Introduction to Programming in Java
  Addison-Wesley, 2007 (ISBN: 0-321-49805-4)

Exams

Organization of the exams:

- Two written tests, scheduled at the end of each semester;
- A few simple laboratory assignments;
- Oral discussion.

Further Information

Further digital material, including programs' code, is available through the course online pages:

https://users.dimi.uniud.it/~claudio.mirolo/teaching/programmazione/

FOUNDATIONS of COMPUTER PROGRAMMING

(teachers/lecturers: Claudio Mirolo and Dario Della Monica)

Teaching Approach

Theoretical lessons and practical laboratory work.

The exams are mainly focused on the student's ability to understand small-scale programs and to design and organize algorithmic solutions to simple problems.

Dublin Descriptors

1. "HARD SKILLS"

1.1. Knowledge and understanding

- Basic knowledge of functional and imperative programming;
- First rudiments of of object-oriented programming;
- Basic knowledge of the syntax and semantics of the main program and data structures in Scheme and Java;
- Knowledge of the relationships between recursion and iteration;
- Knowledge of the relationships between the concepts of class and object.

1.2 Applying knowledge and understanding

- Being able to code simple recursive and imperative algorithms;
- Being able to analyze the logic of a simple recursive or imperative program in order to verify its correctness with respect to the given specifications as well as to estimate its performances;
- Being able to apply the memoization and dynamic programming techniques to improve the performances of recursive programs;
- Being able to implement a class with a given interface.

2. "SOFT SKILLS"

2.1 Making judgements

- Being able to analyze problems in order to identify what can be achieved by a program, to formalize appropriate specifications and to choose appropriate programming tools;
- Being able to understand information-processing processes.

2.2 Communication skills

- Being able to work together with peers in order to design or improve the algorithmic solution of a problem.

2.3 Learning skills

- Being able to plan experiments on a program to test its correctness and to estimate its performances;
- Being able to learn autonomously new programming languages.