Teaching *Programming Techniques*: Methods and Experiences

Costin Bădică, Alex Becheru

Department of Computers and Information Technology

University of Craiova, Romania

Talk Outline

- Course Overview
- Practical Aspects
- Project
- Sample Exam Exercises
- Educational Experiences
- Conclusions

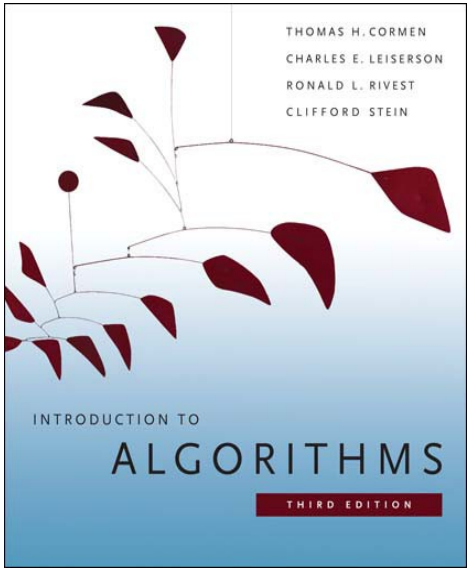
Overview

- Programming Techniques (PT)
  - Planned to introduce students to the methods and techniques of programming and experimenting with fundamental algorithms.
  - Alignment with CS curricula recommended by ACM and IEEE
  - 1st year, 2nd semester
  - Courses that must be passed before PT:
    - Computer Programming
  - Courses that benefit from PT:
    - Object-Oriented Programming
    - Data Structures and Algorithms

Overview – Learning Objectives

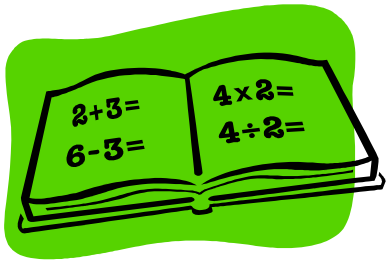
- **LO1:** *To introduce the principles of algorithm analysis, modular programming and data abstraction.*
- **LO2:** *To introduce fundamental algorithms and methods of algorithm design.*
- **LO3:** *To develop practical experience in applying PT for small-scale experiments involving the implementation, testing and evaluation of algorithms.*

Overview – Structure



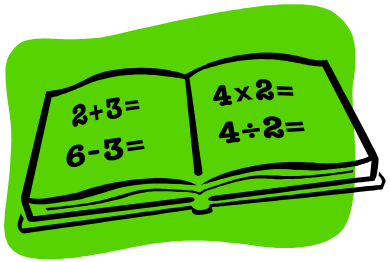
- No single textbook, although a good base is the CLRS3 book.
- 2 modules:
  - Course (4 ECTS - European Credit Transfer and Accumulation System points)
  - Project (1 ECTS points)
- Both have a duration of 14 weeks:
  - Course: 2h lectures/week (28h) + 2 h lab/week (28h - mandatory)
  - Project: 1h project/week (14h)

Overview – Topics I



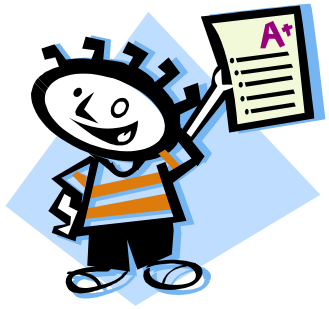
- Introduction to analysis and design of algorithms
- Divide and Conquer
- Correctness and testing of algorithms
- Sorting algorithms
- Abstract data types
- Stacks and queues

Overview – Topics II



- Introduction to graphs and trees
- Graph search and traversal
- Dynamic programming
- Greedy algorithms
- Backtracking
- Introduction to NP-completeness

Overview – Grading



- Course module: final exam (70%)
  - Exercise: discuss, analyze and improve a simple algorithm
  - Exercise: design and code a small-scale C program for solving an algorithmic problem
  - Exercise: design an algorithm using a method
- Course module: laboratory assignments (30%)
- Project module: project assignment
  - 20% intermediary delivery
  - 80% final delivery

Practical Aspects – Programming Language

- We are using ***Standard C***
- Reasons:
  - Students learn C during the 1st semester at the Computer Programming course
  - C is a basis for learning other C-like languages: C++, Java, C#
  - C is sometimes defined as a *high-level assembly language*, directly useful for developing:
    - Operating systems
    - Embedded systems
    - Compilers
  - C encourages very efficient implementation of algorithms
  - PT is a good opportunity for students to strengthen their knowledge of C

Practical aspects – Development Tools

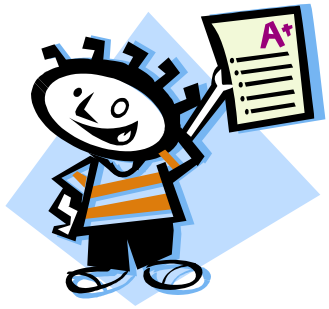
- We are using Code::Blocks
- But we also encourage students to develop C programs using standard Unix tools:
  - GCC
  - Makefiles

Practical aspects – Coding Style

```
typedef struct list_node {
    struct list_node *next;
    struct list_node *prev;
    int key;
} ListNode;
```

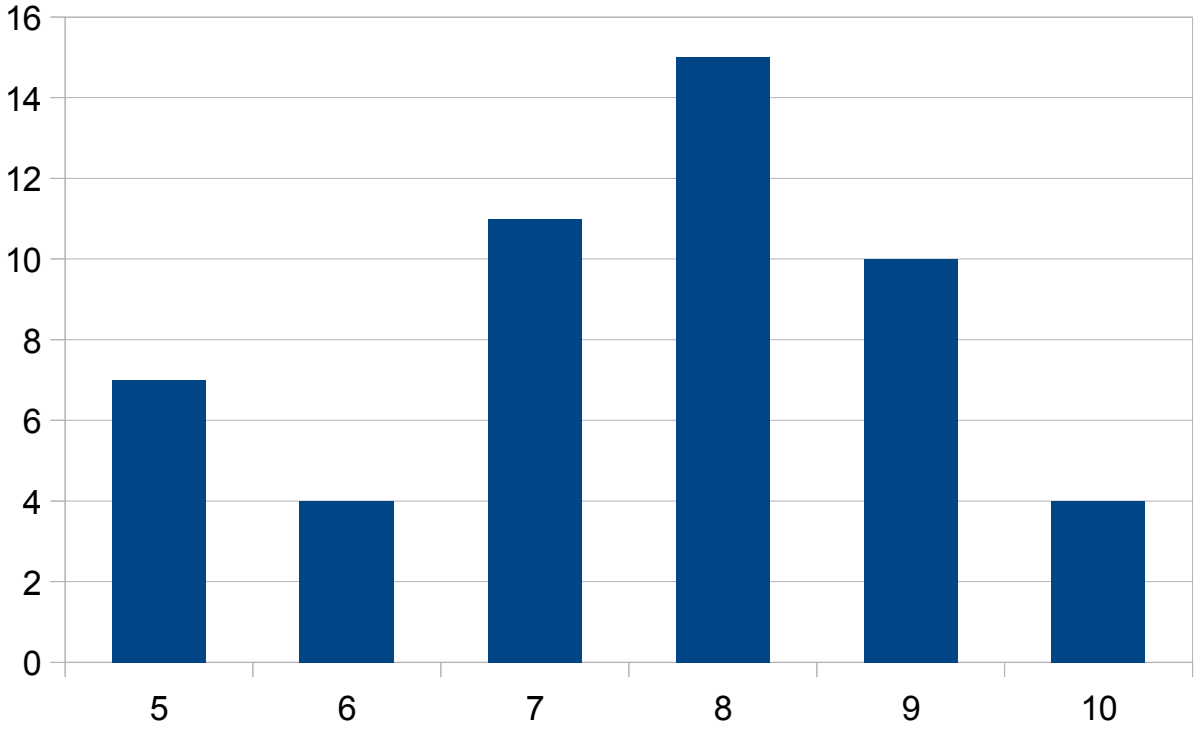
- Use indentation
- Use meaningful names for variables
- Comment your program

Project – Deliverables



- Technical report
  - Describes work and achievements
- Source code
  - Sources, binaries, makefiles, documentation
- Experimental data and results
  - Non-trivial input-output data sets

Results - Grades



- 2014 – 2015 Summer Session
- 108 students, 51 passed, 37 failed, 20 absent

Educational Experiences – Lectures I

- Mixing *theoretical aspects* (algorithms, correctness proofs, algorithms analysis) with *practical aspects* (sample C programs) is a positive thing.
- Each lecture contains at least one practical example of coding an algorithm by highlighting (and repeating):
  - Coding style
  - Comments
  - Modularity
  - Test data

Educational Experiences – Lectures II

- Homework for challenging students and keeping them in the loop:
  - Problem statements ⇒ students must design the algorithm
  - Problem statement + half-baked-algorithm ⇒ students must finalize the algorithm design

Educational Experiences – Lectures / Difficulties I

- Lectures attendance is not mandatory ⇒ it is difficult to motivate the students attendance
  - They have the textbook and the lecture slides
  - We should set more emphasis on applications (problems, exercises, examples) and practical aspects
- Students do not like theory so much.
  - Analysis of algorithms complexity
  - Proving correctness
  - Sometimes these are really difficult tasks

Educational Experiences – Lectures / Difficulties II

- Confusion between (i) understanding an algorithm and (ii) running it to check that it executes correctly, i.e. it produces the correct output (code + test)
- Students do not realize the importance of correctly understanding algorithms. Their argument is that most often in practice algorithms are already efficiently implemented in libraries and available via APIs. But:
  - They are also a tool for developing an "algorithmic thinking", useful for computer scientists.
  - Sometimes you have to implement, update, or select an algorithm for a given problem.

Educational Experiences – Laboratories

- Programming exercises
  - Algorithm ⇒ program code
  - Preparation of sets of input-output tests
- Difficulties
  - Students perceive C as a difficult programming language
  - Students avoid (and minimize the importance of) the development of their programs using command-line tools: GCC and *make*
  - Students minimize the importance of preparing good test data
  - Very often they ask why do we not use GUIs, missing the point that PT is not about GUI design and programming

Educational Experiences – Project / Difficulties I

- Block diagram
  - Confusion between the inputs and outputs of the program, the components (modules) of the program, the source files (.*h* and .*c*), and the program C functions.
- Pseudocode:
  - Students tend to rewrite the whole C program in pseudocode, rather than to focus only on the algorithm description in pseudocode.
  - Students are doing the pseudocode after they are coding the C program

Educational Experiences – Project / Difficulties II

- Source code
  - Not commented
  - Does not follow a coding style
  - Mix of C and C++ constructs
  - Failing to compile using command-line tools
  - Minimizing the importance of testing by failing to provide non-trivial input data tests
  - Difficulties in designing and preparing input data tests. Actually, they start thinking at input data tests after coding a large part of the program. This determines redundant work.

Conclusions

- Programming Techniques proved to be a useful course that acts as glue between:
  - Introductory programming (1st year, 1st semester)
  - More advanced courses, Algorithms and Data Structures and Software Engineering (2nd and 3rd years)
- (Some) students start to understand that theory and practice of programming go hand-in-hand:
  - Better analyzed and understood problems $\Rightarrow$ programs are easier to develop, more efficient and easier to maintain.


Thank You