Course guide
270005 - PRO2 - Programming II

Unit in charge: Barcelona School of Informatics
Teaching unit: 723 - CS - Department of Computer Science.

Degree: BACHELOR’S DEGREE IN INFORMATICS ENGINEERING (Syllabus 2010). (Compulsory subject).

Academic year: 2022  ECTS Credits: 7.5  Languages: Catalan, Spanish

LECTURER

Coordinating lecturer: RENATO ALQUEZAR MANCHO - BORJA VALLES FUENTE - JUAN LUIS ESTEBAN ÁNGELES

Others: Primer quadrimestre:
M. LUISA BONET CARBONELL - 13, 41
JUAN LUIS ESTEBAN ÁNGELES - 11, 12, 13
XAVIER MESSEGUER PEYPOCH - 11, 43
BORJA VALLES FUENTE - 12, 41, 42, 43

PRIOR SKILLS

Students are expected to have been trained in imperative programming techniques based on:
- basic instructions (assignment, alternative and iteration)
- actions and functions, parameter passing and recurrence
- vectors, tuples and sequences
- sequential search and traversal schemes
- basic algorithms (binary search, vector sorting, matrix arithmetics).

They are also expected to know how to use at least one imperative language, preferably C++, and they should have some experience in implementing of C++ programs in the Linux environment.

Furthermore, they should be able to assimilate information from a statement, discuss the correctness of an algorithm and compare algorithmic solutions.
DEGREE COMPETENCES TO WHICH THE SUBJECT CONTRIBUTES

Specific:
CT1.1A. To demonstrate knowledge and comprehension about the fundamentals of computer usage and programming, about operating systems, databases and, in general, about computer programs applicable to the engineering.
CT1.1B. To demonstrate knowledge and comprehension about the fundamentals of computer usage and programming. Knowledge about the structure, operation and interconnection of computer systems, and about the fundamentals of its programming.
CT1.2B. To interpret, select and value concepts, theories, uses and technological developments related to computer science and its application derived from the needed fundamentals of mathematics, statistics and physics. Capacity to understand and dominate the physical and technological fundamentals of computer science: electromagnetism, waves, circuit theory, electronics and photonics and its application to solve engineering problems.
CT3.6. To demonstrate knowledge about the ethical dimension of the company: in general, the social and corporative responsibility and, concretely, the civil and professional responsibilities of the informatics engineer.
CT4.1. To identify the most adequate algorithmic solutions to solve medium difficulty problems.
CT4.2. To reason about the correction and efficiency of an algorithmic solution.
CT5.1. To choose, combine and exploit different programming paradigms, at the moment of building software, taking into account criteria like ease of development, efficiency, portability and maintainability.
CT5.2. To know, design and use efficiently the most adequate data types and data structures to solve a problem.
CT5.3. To design, write, test, refine, document and maintain code in an high level programming language to solve programming problems applying algorithmic schemas and using data structures.
CT5.4. To design the programs’ architecture using techniques of object orientation, modularization and specification and implementation of abstract data types.
CT8.6. To demonstrate the comprehension of the importance of the negotiation, effective working habits, leadership and communication skills in all the software development environments.

Generical:
G5. TEAMWORK: to be capable to work as a team member, being just one more member or performing management tasks, with the finality of contributing to develop projects in a pragmatic way and with responsibility sense; to assume compromises taking into account the available resources.

TEACHING METHODOLOGY

Topics will be explained in a practical way by using many examples.

Theory classes introduce knowledge, techniques and concepts that will be used in laboratory sessions. They also include the presentation and discussion of the solutions of a set of problems.

The two-hour theory classes and the three-hour laboratory sessions will take place weekly.

The programming project integrates knowledge and skills of the entire course, except for the topic (recursive data types) which will be assessed in the final theory exam.
LEARNING OBJECTIVES OF THE SUBJECT

1. To distinguish the roles of user, specifier and implementer of data classes. To know the main components of the specification of a class of data. To know the main components of the implementation of a class of data.
2. To design a class of data with a clear independence between specification and implementation. To justify why an object of the class can only be created, consulted or modified using the operations in the class specification.
3. To solve any exercise which requires the application of a simple algorithm to a vector of objects of a class of data in C++.
4. Given an implementation for a simple class of data, make improvements in its representation and its operations.
5. To explain the main stages of modular design.
6. To identify in a textual statement of a problem data abstractions that may be represented by classes of data which could be used to solve the problem. To check whether any of the abstractions identified has been previously detected, in order to reuse the corresponding class of data.
7. To individually design a modular program in C++ from the data abstractions identified by analysing the statement of a problem. To modify or add some functionality to a given modular program written in C++.
8. To implement a modular program in C++ elegantly and in such a way that other programmers can understand what it does and modify it. To write documentation that facilitates the use of a modular program written in C++ by other programmers.
9. Prepare a C++ program which uses simple data types and C++ classes (some of the predefined and others defined by the student) to be executed. Thw student should be able to do this in two ways: 1) compiling and linking the program using the g++ command; and 2) writing a makefile file, and using it to compile and link the program.
10. To design in teams a modular program in C++ and/or a set of cases (i.e. a set of examples of correct input and corresponding output) to test the full functionality of the program. To debug this program systematically, so that small implementation errors are removed in a reasonable period of time.
11. To know the data types typically used to represent and manage linear data structures and their specification. To design iterative and recursive algorithms for solving search and traversal problems on stacks, queues and lists, using the operations of the corresponding data type and iterators (when appropriated).
12. To know data types used to represent and manage tree data structures and their specification. To design recursive algorithms for solving search and traversal problems about binary, n-ary and general trees, using the operations of the corresponding data type.
13. To describe the main steps in the design of iterative algorithms. To justify the correctness of relatively simple iterative algorithms.
14. To describe the main steps in the design of recursive algorithms. To justify the correctness of relatively simple recursive algorithms.
15. To know what a generalization of a function is, and to be able to explain the difference between specification generalizations and efficiency generalizations. To know the different types of specification generalizations and the different types of efficiency generalizations.
16. Given a simple recursive algorithm, to determine whether there is a straightforward way to obtain an equivalent iterative algorithm, and write it if there exists such a straightforward transformation.
17. To distinguish whether the cost of a given iterative or recursive simple algorithm which works on vectors, stacks, queues, lists or trees is linear or if it is quadratic (assuming that the cost is of one of these two types).
18. To determine if the efficiency of a given simple recursive algorithm can be improved and, if it is possible, to design a more efficient recursive algorithm that solves the same problem using efficiency generalizations.
19. To determine if the efficiency of a given simple iterative algorithm can be improved and, if it is possi

STUDY LOAD

<table>
<thead>
<tr>
<th>Type</th>
<th>Hours</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours small group</td>
<td>45,0</td>
<td>24.39</td>
</tr>
<tr>
<td>Self study</td>
<td>102,0</td>
<td>55.28</td>
</tr>
<tr>
<td>Hours large group</td>
<td>30,0</td>
<td>16.26</td>
</tr>
<tr>
<td>Guided activities</td>
<td>7,5</td>
<td>4.07</td>
</tr>
</tbody>
</table>

Total learning time: 184.5 h
### Modular design and object-oriented design

**Description:**

Basic principles of object-oriented design: classes and objects; fields and methods.

Implementing modular designs in C++. Separate compilation and linking. Debugging, testing and documentation of modular programs.

### Linear data structures

**Description:**
Stacks, queues and lists with a point of interest: specification and use (search and traversal operations). Iterators: definition and use.

### Tree data structures

**Description:**
Binary, n-ary and general trees: specification. Use of binary trees: search and traversal operations.

### Iterative program correctness

**Description:**
Loop invariants. Justification of the correctness of iterative algorithms.

### Recursive programming and correctness of recursive algorithms

**Description:**

### Efficiency enhancements for recursive and iterative programs

**Description:**
Detection of repeated calculations in recursive and iterative programs. Efficiency generalisations: new data (input parameters) and/or results (return values or output parameters) in recursive operations to improve efficiency. New local variables that use their previous values in iterative operations to improve efficiency.

### Recursive data types

**Description:**
Introduction to the use of recursive data types. Pointer type constructor and dynamic memory management. Implementation of linked data structures by means of recursive data types. Iterative and recursive algorithms for solving search and traversal problems in linked data structures by directly accessing the representation based on nodes and node pointers.
## ACTIVITIES

**Introduction to modular design and object-oriented design.**

*Description:* Development of the corresponding topic.

*Specific objectives:* 1, 2, 3

*Full-or-part-time:* 8h
  - Theory classes: 4h
  - Self study: 4h

---

**C++ review exercises.**

*Description:* Link to PRO1 contents.

*Specific objectives:* 3, 9

*Full-or-part-time:* 6h
  - Laboratory classes: 3h
  - Self study: 3h

---

**Specification and use of classes of objects in C++.**

*Description:* Exercises from the corresponding topic.

*Specific objectives:* 1, 2, 3

*Full-or-part-time:* 12h
  - Laboratory classes: 6h
  - Self study: 6h

---

**Implementing classes of objects in C++.**

*Description:* Exercises from the corresponding topic.

*Specific objectives:* 1, 2, 4

*Full-or-part-time:* 6h
  - Laboratory classes: 3h
  - Self study: 3h
### Example of modular design 1

**Description:**
An exercise on the corresponding topic.

**Specific objectives:**
1, 2, 5, 6, 7, 8, 9

**Full-or-part-time:** 6h  
Laboratory classes: 3h  
Self study: 3h

### Linear data structures.

**Description:**
Development of the corresponding topic and laboratory exercises.

**Specific objectives:**
11

**Full-or-part-time:** 12h  
Theory classes: 2h  
Laboratory classes: 4h  
Self study: 6h

### Tree data structures.

**Description:**
Development of the corresponding topic and laboratory exercises.

**Specific objectives:**
12

**Full-or-part-time:** 8h  
Theory classes: 2h  
Laboratory classes: 2h  
Self study: 4h

### Iterative program correctness

**Description:**
Development of the corresponding topic.

**Specific objectives:**
13

**Full-or-part-time:** 4h  
Theory classes: 2h  
Self study: 2h
Example of modular design 2

Description:
An exercise on the corresponding topic.

Specific objectives:
1, 2, 5, 6, 7, 8, 9

Full-or-part-time: 6h
Laboratory classes: 3h
Self study: 3h

Delivery of the specification for the practical

Description:
The students will deliver a document with the specification of their solution to the practical.

Specific objectives:
1, 2, 5, 6, 7, 8

Full-or-part-time: 9h
Laboratory classes: 3h
Self study: 6h

Recursive programming

Description:
Development of the corresponding topic.

Specific objectives:
14, 15, 16

Full-or-part-time: 6h
Theory classes: 3h
Self study: 3h

Example of modular design 3 + Review of the specification for the practical

Description:
An exercise on the corresponding topic.

Specific objectives:
1, 2, 5, 6, 7, 8, 9

Full-or-part-time: 8h
Laboratory classes: 3h
Self study: 5h
Efficiency enhancements for recursive and iterative programs.

Description:
Development of the corresponding topic.

Specific objectives:
13, 14, 15, 16, 17, 18, 19

Full-or-part-time: 6h
Theory classes: 3h
Self study: 3h

Supervision of practical 1

Description:
Supervision of the design and implementation of the practical.

Specific objectives:
1, 2, 4, 6, 7, 8, 9, 11, 12, 17, 18

Full-or-part-time: 10h
Laboratory classes: 3h
Self study: 7h

Supervision of practical 2

Description:
Supervision of the design and implementation of the practical.

Specific objectives:
1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 17, 18

Related competencies:
G5. TEAMWORK: to be capable to work as a team member, being just one more member or performing management tasks, with the finality of contributing to develop projects in a pragmatic way and with responsibility sense; to assume compromises taking into account the available resources.

Full-or-part-time: 10h
Laboratory classes: 3h
Self study: 7h

Supervisión del diseño e implementación de la práctica.

Description:
Supervision of the design and implementation of the practical.

Specific objectives:
1, 2, 4, 6, 7, 8, 9, 11, 12, 17, 18, 19

Full-or-part-time: 6h
Laboratory classes: 3h
Self study: 3h
Recursive data types.

Description:
Development of the corresponding topic.

Specific objectives:
2, 4, 17, 18, 19, 20, 22

Full-or-part-time: 32h
Theory classes: 10h
Laboratory classes: 6h
Self study: 16h

Review of theory and exam problems.

Description:
Questions can be asked about the topics covered in theory classes.

Full-or-part-time: 4h
Theory classes: 2h
Self study: 2h

Consolidation

Description:
Consolidation (week 15)

Specific objectives:
11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22

Full-or-part-time: 8h
Theory classes: 2h
Self study: 6h

ESP_PRAC

Description:
Specification of the most important classes and main program of the programming project.

Specific objectives:
1, 2, 5, 6, 7, 8

Full-or-part-time: 1h
Self study: 1h
**EX_PAR1_TP**

**Description:**
Mid-term theory/problems exam.

**Specific objectives:**
1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19

**Full-or-part-time:** 4h
Guided activities: 2h
Self study: 2h

---

**CODI_DOC_PRAC1**

**Description:**
Delivery of the programming project code and documentation.

**Specific objectives:**
1, 2, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19

**Related competencies:**
G5. TEAMWORK: to be capable to work as a team member, being just one more member or performing management tasks, with the finality of contributing to develop projects in a pragmatic way and with responsibility sense; to assume compromises taking into account the available resources.

**Full-or-part-time:** 4h
Guided activities: 1h
Self study: 3h

---

**EX_PRAC**

**Description:**
Programming project exam.

**Specific objectives:**
1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19

**Full-or-part-time:** 4h 30m
Guided activities: 2h 30m
Self study: 2h

---

**EX_PAR2_TP**

**Description:**
Final theory/problems exam.

**Specific objectives:**
1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22

**Full-or-part-time:** 4h
Guided activities: 2h
Self study: 2h
GRADING SYSTEM

The technical competence mark (NCTEC) is calculated as follows:

\[ NCTEC = 0.20 \times \text{EX\_PAR1\_TP} + 0.35 \times \text{PRAC} + 0.15 \times \text{EX\_PRAC} + 0.30 \times \text{EX\_PAR2\_TP} \]

where

* \text{EX\_PAR1\_TP} and \text{EX\_PAR2\_TP} are the marks of the mid-term and final theory exams
* \text{PRAC} is the mark of the programming project; it may have an automatic part, derived from code testing, and a manual part
* \text{EX\_PRAC} is the mark of the programming project exam; to be able to take this exam, the automatic part (if it exists) of mark PRAC needs to be greater than 0

Nevertheless, NCTEC will be NP if the weight of the assessment activities with an NP mark is greater than or equal to 70%.

The assessment of the generic competence Teamwork will be carried out as follows. In a previously appointed laboratory session, teams of students will be formed. Each team should generate and deliver a number of cases (i.e. a set of examples consisting of pairs of correct input and the corresponding output) for testing the full functionality of the programming project C++ code. The evaluation of the competence Teamwork for each student in a team will be based on a document delivered by the team which will contain a description of the test cases generated and a detailed explanation of the participation of each team member in their development.

BIBLIOGRAPHY

Basic:
- Alquezar, René; Valles, Borja; [et al.]. Apunts de teoria de PRO2.
- Valles, Borja; [et al.]. Sessions de Laboratori de PRO2.

Complementary:

RESOURCES

Hyperlink:
- [http://www.cs.upc.edu/~pro2](http://www.cs.upc.edu/~pro2)