270005 - PRO2 - Programming II

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

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.
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. The 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.

| Study load |  |
| --- | --- | --- | --- |
| Total learning time: 184h 30m | Hours large group: 30h | 16.26% |
|  | Hours medium group: 0h | 0.00% |
|  | Hours small group: 45h | 24.39% |
|  | Guided activities: 7h 30m | 4.07% |
|  | Self study: 102h | 55.28% |
## Modular design and object-oriented design

**Degree competences to which the content contributes:**

**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

**Degree competences to which the content contributes:**

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

## Tree data structures

**Degree competences to which the content contributes:**

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

## Iterative program correctness

**Degree competences to which the content contributes:**

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

## Recursive programming and correctness of recursive algorithms

**Degree competences to which the content contributes:**

**Description:**
### Efficiency enhancements for recursive and iterative programs

**Degree competences to which the content contributes:**

**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

**Degree competences to which the content contributes:**

**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.
### Planning of activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Hours</th>
<th>Theory classes</th>
<th>Practical classes</th>
<th>Laboratory classes</th>
<th>Guided activities</th>
<th>Self study</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction to modular design and object-oriented design.</strong></td>
<td>8h</td>
<td>4h</td>
<td>0h</td>
<td>0h</td>
<td>0h</td>
<td>4h</td>
</tr>
<tr>
<td><strong>C++ review exercises.</strong></td>
<td>6h</td>
<td>0h</td>
<td>0h</td>
<td>0h</td>
<td>0h</td>
<td>3h</td>
</tr>
<tr>
<td><strong>Specification and use of classes of objects in C++.</strong></td>
<td>12h</td>
<td>0h</td>
<td>0h</td>
<td>6h</td>
<td>0h</td>
<td>6h</td>
</tr>
<tr>
<td><strong>Implementing classes of objects in C++.</strong></td>
<td>6h</td>
<td>0h</td>
<td>0h</td>
<td>3h</td>
<td>0h</td>
<td>3h</td>
</tr>
</tbody>
</table>

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

**Specific objectives:**
1, 2, 3

**Description:**
Link to PRO1 contents.

**Specific objectives:**
3, 9

**Description:**
Exercises from the corresponding topic.

**Specific objectives:**
1, 2, 3
### Example of modular design 1

#### Description:
Exercises from the corresponding topic.

#### Specific objectives:
1, 2, 4

#### Hours:
- Theory classes: 0h
- Practical classes: 0h
- Laboratory classes: 3h
- Guided activities: 0h
- Self study: 3h

### Linear data structures.

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

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

#### Hours:
- Theory classes: 2h
- Practical classes: 0h
- Laboratory classes: 4h
- Guided activities: 0h
- Self study: 6h

### Tree data structures.

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

#### Specific objectives:
11

#### Hours:
- Theory classes: 2h
- Practical classes: 0h
- Laboratory classes: 2h
- Guided activities: 0h
- Self study: 4h
<table>
<thead>
<tr>
<th>Topic</th>
<th>Hours</th>
<th>Description</th>
<th>Specific objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative program correctness</td>
<td>4h</td>
<td>Development of the corresponding topic.</td>
<td>13</td>
</tr>
<tr>
<td>Example of modular design 2</td>
<td>6h</td>
<td>An exercise on the corresponding topic.</td>
<td>1, 2, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>Delivery of the specification for the practical</td>
<td>9h</td>
<td>The students will deliver a document with the specification of their solution to the practical.</td>
<td>1, 2, 5, 6, 7, 8</td>
</tr>
<tr>
<td>Recursive programming</td>
<td>6h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Description:
Development of the corresponding topic.

### Specific objectives:
- 14, 15, 16

---

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

<table>
<thead>
<tr>
<th>Hours: 8h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory classes: 0h</td>
</tr>
<tr>
<td>Practical classes: 0h</td>
</tr>
<tr>
<td>Laboratory classes: 3h</td>
</tr>
<tr>
<td>Guided activities: 0h</td>
</tr>
<tr>
<td>Self study: 5h</td>
</tr>
</tbody>
</table>

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

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

---

### Efficiency enhancements for recursive and iterative programs.

<table>
<thead>
<tr>
<th>Hours: 6h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory classes: 3h</td>
</tr>
<tr>
<td>Practical classes: 0h</td>
</tr>
<tr>
<td>Laboratory classes: 0h</td>
</tr>
<tr>
<td>Guided activities: 0h</td>
</tr>
<tr>
<td>Self study: 3h</td>
</tr>
</tbody>
</table>

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

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

---

### Supervision of practical 1

<table>
<thead>
<tr>
<th>Hours: 12h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory classes: 0h</td>
</tr>
<tr>
<td>Practical classes: 0h</td>
</tr>
<tr>
<td>Laboratory classes: 3h</td>
</tr>
<tr>
<td>Guided activities: 0h</td>
</tr>
<tr>
<td>Self study: 9h</td>
</tr>
</tbody>
</table>

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

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

**Hours:** 10h
- Theory classes: 0h
- Practical classes: 0h
- Laboratory classes: 3h
- Guided activities: 0h
- Self study: 7h

---

### Recursive data types.

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

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

**Hours:** 32h
- Theory classes: 10h
- Practical classes: 0h
- Laboratory classes: 6h
- Guided activities: 0h
- Self study: 16h

---

### Review of theory and exam problems.

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

**Hours:** 4h
- Theory classes: 2h
- Practical classes: 0h
- Laboratory classes: 0h
- Guided activities: 0h
- Self study: 2h

---

### Consolidation

**Description:**
Consolidation (week 15)

**Hours:** 11h
- Theory classes: 2h
- Practical classes: 0h
- Laboratory classes: 3h
- Guided activities: 0h
- Self study: 6h
### C

**Description:**
Mid-term laboratory exam

**Specific objectives:**
1, 2, 3, 4, 8, 9

<table>
<thead>
<tr>
<th>Hours</th>
<th>Guided activities</th>
<th>Self study</th>
</tr>
</thead>
<tbody>
<tr>
<td>2h</td>
<td>1h</td>
<td>1h</td>
</tr>
</tbody>
</table>

### ESP_PRAC

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

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

<table>
<thead>
<tr>
<th>Hours</th>
<th>Guided activities</th>
<th>Self study</th>
</tr>
</thead>
<tbody>
<tr>
<td>1h</td>
<td>0h</td>
<td>1h</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Hours</th>
<th>Guided activities</th>
<th>Self study</th>
</tr>
</thead>
<tbody>
<tr>
<td>4h</td>
<td>2h</td>
<td>2h</td>
</tr>
</tbody>
</table>

### CODI_DOC_PRAC

**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

<table>
<thead>
<tr>
<th>Hours</th>
<th>Guided activities</th>
<th>Self study</th>
</tr>
</thead>
<tbody>
<tr>
<td>2h</td>
<td>0h</td>
<td>2h</td>
</tr>
</tbody>
</table>
The technical competence mark (NCTEC) is calculated as follows:

$$NCTEC = 0.10 \times C + 0.25 \times EX\_PAR1\_TP + 0.15 \times EX\_PRAC + 0.25 \times PRAC + 0.25 \times EX\_PAR2\_TP$$

where

* $C$ is the mark of the mid-term laboratory exam; to be able to take this exam a subset of the exercises of the first three sessions of laboratory must be solved.

* $EX\_PAR1\_TP$ and $EX\_PAR2\_TP$ are the marks of the mid-term and final theory exams

* $EX\_PRAC$ is the mark of the programming project exam

* $PRAC$ is the mark of the programming project

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, R.; Valles, B. [et al.]. Apuntes de teoría de PRO2.

Valles, B. [et al.]. Sesiones de Laboratorios de PRO2.


Complementary:


Others resources:

Hyperlink

http://www.cs.upc.edu/~pro2

http://c.conclase.net/curs/index.php

http://wwwcplusplus.com/reference/stl/