340371 - PRO1-I2O23 - Programming I

Coordinating unit: 340 - EPSEVG - Vilanova i la Geltrú School of Engineering
Teaching unit: 723 - CS - Department of Computer Science
Academic year: 2018
Degree: BACHELOR’S DEGREE IN INFORMATICS ENGINEERING (Syllabus 2010). (Teaching unit Compulsory)
ECTS credits: 7.5
Teaching languages: Catalan

Teaching staff
Coordinator: Jordi Esteve Cusiné, Neus Català Roig
Others: Jordi Esteve Cusiné, Neus Català Roig, Ignasi Gómez Sebastià

Opening hours

Timetable: See the current timetable in the EPSEVG people list:
https://web3.epsevg.upc.edu/coneix-lepsevg/directori-epsevg

Prior skills

Required knowledge on imperative programming techniques:
- basic instructions: assignment, alternative, iteration
- actions, functions and parameter passing
- vectors, tuples and sequences
- sequential search and traversal schemes

Students are expected to know how to use one imperative language, preferably C++. They should have some experience in editing, compiling and running C++ programs in the Linux environment.

Requirements

Have passed FOPR or at least being enrolled.

Degree competences to which the subject contributes

Specific:
1. CEFB3. Ability to understand and to have a good command of discrete, logical, algorithmically mathematics and computing complexity and its application to automatical treatment of information by means of computational systems and its application to solve engineering problems.
2. CEFB4. Basic knowledge of use and computer programming, as well as of operating systems, data base and generally informatic programs with engineering applications.
3. CEFB5. Knowledge of informatic systems, its structure, function and interconnection, as well as fundamentals of its programming.
4. CEFC6. Basic knowledge and application of algorithmic processes, informatic techniques to design solutions of problems, analyzing if proposed algorsims are apt and complex.
5. CEFC7. Knowledge, design and efficient use of data types and structures the most appropriate to resolve problems.

Transversal:
6. SELF-DIRECTED LEARNING - Level 1. Completing set tasks within established deadlines. Working with recommended information sources according to the guidelines set by lecturers.
The objective of this course is to consolidate the basic techniques of designing algorithms for solving problems by computer, scientific and technical fields, and learn the basics of advanced techniques such as recursivity and object orientation.

After completing the course the student has to:
- Proficiency about the concepts of class, object, attribute, method, and understand class specifications.
- Build programs that use classes for simple linear structures (stack, queue, list, vector) and tree (binary tree, general tree).
- Understanding of the multiple and linear recursivity and their relationship with the iterative algorithms.
- Design correct and efficient programs both iterative and recursive.
- Explain the different types of specification generalizations and their characteristics.
- Be able to implement a data structure with specific operational and efficiency requirements using recursive data types (or pointers).

Teaching methodology

The methodological approach consists of:
- Presentation in the classroom, participatory classes, concepts and procedures associated with the subject (2 hours a week).
- Problem solving, individually or in teams, presentially.
- Problem solving, individually or in teams as non-presential activity.
- Computer lab work, individually or in teams, presentially (3 hours a week).
- Computer lab work, individually or in teams, non-presentially.
- Individual tests and exams, presentially.

Learning objectives of the subject

The objective of this course is to consolidate the basic techniques of designing algorithms for solving problems by computer, scientific and technical fields, and learn the basics of advanced techniques such as recursivity and object orientation.

After completing the course the student has to:
- Proficiency about the concepts of class, object, attribute, method, and understand class specifications.
- Build programs that use classes for simple linear structures (stack, queue, list, vector) and tree (binary tree, general tree).
- Understanding of the multiple and linear recursivity and their relationship with the iterative algorithms.
- Design correct and efficient programs both iterative and recursive.
- Explain the different types of specification generalizations and their characteristics.
- Be able to implement a data structure with specific operational and efficiency requirements using recursive data types (or pointers).

Study load

<table>
<thead>
<tr>
<th>Total learning time: 187h 30m</th>
<th>Hours large group: 30h 16.00%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours medium group: 0h 0.00%</td>
<td></td>
</tr>
<tr>
<td>Hours small group: 45h 24.00%</td>
<td></td>
</tr>
<tr>
<td>Guided activities: 0h 0.00%</td>
<td></td>
</tr>
<tr>
<td>Self study: 112h 30m 60.00%</td>
<td></td>
</tr>
</tbody>
</table>
## Content

### 1.- Modular design and object-oriented design

**Learning time:** 20h  
- Theory classes: 2h  
- Practical classes: 3h  
- Laboratory classes: 2h  
- Guided activities: 1h  
- Self study: 12h

**Description:**  

**Related activities:**  
Activity 1 and First Test.

**Specific objectives:**  
- Distinguish the roles of user, specifier and implementer of data classes. List the elements of the specification of a data class. List the elements of the implementation of a data class.
- Design a data class with a clear independence between specification and implementation. Justify why the only way to create, consult or modify an object of a data class is through operations included in the class specification.
- Solve in C++ any exercise based on the application of a basic algorithmic technique on a vector composed of objects from a data class, as would be done for a vector of elements from simple types.
- Given an implementation for a simple data class, make improvements in the representation of its components and its operations.

### 2.- Linear and tree-like data structures

**Learning time:** 26h  
- Theory classes: 3h  
- Practical classes: 4h  
- Laboratory classes: 2h  
- Guided activities: 1h  
- Self study: 16h

**Description:**  

**Related activities:**  
Activity 2, First Test and Final Test.

**Specific objectives:**  
- Identify the data types most used to represent and manage linear data structures and write their specification. Design iterative and recursive algorithms for solving search and traversal problems on stacks, queues and lists, using the operations of the corresponding data types.
- Identify the data types most used to represent and manage tree data structures and write their specification. Design recursive algorithms for solving search and traversal problems on binary and general trees using the operations of the corresponding data type.
3.- Methodical recursive programming

**Description:**
Inductive design of recursive algorithms. Proof of correctness of recursive algorithms. Function immersion (or generalization). Relationship between final linear recursive algorithms and iterative algorithms.

**Related activities:**
First Test, Final Test and Programming project.

**Specific objectives:**
- Describe the steps to follow to design a recursive function.
- Prove the correctness of a given recursive algorithm.
- Explain what a function immersion is and the difference between specification and efficiency immersions.
- Explain the two types of specification immersions and their characteristics.
- Given a recursive algorithm, determine whether there is a simple way to obtain an equivalent iterative algorithm, and if so, write it.

**Learning time:** 39h 30m
- Theory classes: 4h
- Practical classes: 6h
- Laboratory classes: 4h
- Guided activities: 1h
- Self study: 24h 30m

---

4.- Methodical iterative programming

**Description:**
Loop invariants. Inductive design of iterative algorithms. Proof of correctness of iterative algorithms.

**Related activities:**
First Test.

**Specific objectives:**
- Describe the steps to follow to design an iterative algorithm.
- Prove the correctness of a given iterative algorithm.

**Learning time:** 24h 30m
- Theory classes: 2h
- Practical classes: 3h 30m
- Laboratory classes: 3h
- Guided activities: 1h
- Self study: 15h
5.- Efficiency enhancements for recursive and iterative programs

| Learning time: 20h |
| Theory classes: 2h |
| Practical classes: 3h |
| Laboratory classes: 2h |
| Guided activities: 1h |
| Self study : 12h |

**Description:**
Detection of repeated calculations in recursive and iterative programs. Efficiency immersions: new parameters and/or results in recursive operations to improve efficiency. New local variables that reuse their previous value in iterative operations to improve efficiency.

**Related activities:**
Final Test and Programming project.

**Specific objectives:**
- Distinguish whether the cost of a given iterative or recursive algorithm, which works on vectors, stacks, queues or trees, is linear or quadratic (assuming that the cost is one of those).
- Identify whether you can improve the efficiency of a given recursive algorithm and, if possible, design a more efficient recursive algorithm using efficiency immersions.
- Identify whether you can improve the efficiency of a given iterative algorithm and, if possible, design a more efficient alternative iterative algorithm.

6.- Recursive data types

| Learning time: 20h |
| Theory classes: 2h |
| Practical classes: 3h |
| Laboratory classes: 2h |
| Guided activities: 1h |
| Self study : 12h |

**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 types (nodes). Iterative and recursive algorithms for solving search and traversal problems in linked data structures by directly accessing the node-based representation.

**Related activities:**
Final Test.

**Specific objectives:**
- Implement a data structure with specific operational and efficiency requirements using recursive data types (or pointers).
- Design iterative and recursive algorithms for resolving search and traversal problems in linked data structures, by directly accessing the implementation (given) of the appropriate type.
Qualification system

C1 = First Test. Individual written test (2 hours).
C2 = Final Test. Individual written test (maximum 3 hours) which integrates knowledge and skills of the entire course.
Act = Grade obtained from the grade of 2 activities, both activities having the same weight.
Pra = Grade obtained from the grade of each practice delivery (programming project).

Final Grade = max(0,25*C1 + 0,30*C2, 0,55*C2) + 0,10*Act + 0,35*Pra

Practice module:
This module corresponds to the programming project. The students should form teams of two people to solve the proposed programming project which integrates knowledge and skills of the entire course. The assessment of the programming project takes into account the C++ code, the execution of test cases and an Individual Validation Test (PVI). The PVI can be an interview, the follow-up in the laboratory sessions or a short test in the Final Test. The presentation of the programming project will be mandatory to pass the course; otherwise, the Final Grade will be 'NP'.

The Review Test, written test of maximum 3 hours, replaces the grade of the two written tests, therefore corresponds to 55% of the final grade.

Regulations for carrying out activities

The written tests (First Test, Final Test and Review Test) are presential and individual.
The Practical is done in teams of two people. It is delivered non-presentially and evaluated both presentially and non presentially, from the deliverables.

Bibliography

Basic:

Complementary:

Others resources:

Hyperlink
http://c.conclase.net/curso/index
C++ course. To be used as a reference manual.

http://wwwcplusplus.com/reference/stl/
Reference manual for C++ STL containers.