Last update: 05-07-2019

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: 2019

Degree: BACHELOR'S DEGREE IN INFORMATICS ENGINEERING (Syllabus 2018). (Teaching unit Compulsory)
BACHELOR'S DEGREE IN INFORMATICS ENGINEERING (Syllabus 2010). (Teaching unit Compulsory)

ECTS credits: 7,5 Teaching languages: Catalan

Teaching staff

Coordinator: Neus Català Roig, Jordi Esteve Cusiné

Others: Neus Català Roig, Jordi Esteve Cusiné

Opening hours

Timetable: See the current office hours 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, unctions 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 algorisms 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.

7. EFFICIENT ORAL AND WRITTEN COMMUNICATION - Level 1. Planning oral communication, answering questions properly and writing straightforward texts that are spelt correctly and are grammatically coherent.
8. TEAMWORK - Level 1. Working in a team and making positive contributions once the aims and group and individual responsibilities have been defined. Reaching joint decisions on the strategy to be followed.
9. EFFECTIVE USE OF INFORMATI0N RESOURCES - Level 1. Identifying information needs. Using collections, premises and services that are available for designing and executing simple searches that are suited to the topic.

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 tecniques 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

Total learning time: 187h 30m	Hours large group:	30h	16.00%
	Hours medium group:	0h	0.00%
	Hours small group:	45h	24.00%
	Guided activities:	0h	0.00%
	Self study:	112h 30m	60.00%

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:
Abstraction and the need for abstraction. Functional and data decomposition. Modules. Information hiding. Encapsulation. Modular design phases: distinguishing between specification and implementation. Types of modules and their use. Libraries. 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.

Related activities:
Activity 1 and First Test.

Specific objectives:
- 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.
- 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.
- To solve any exercise which requires the application of a simple algorithm to a vector of objects of a class of data in C++.
- Given an implementation for a simple class of data, make improvements in its representation and its operations.
- 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++.
- Prepare a C++ program which uses simple data types and C++ classes (some of them predefined and others defined by the student) to be executed.

2.- Linear and tree data structures	Learning time: 26h Theory classes: 3h Practical classes: 4h Laboratory classes: 2h Guided activities: 1h Self study : 16h

Description:
Stacks, queues and lists: specification and use (search and traversal operations). Iterators: definition and use. Binary, n-ary and general trees: specification and use (search and traversal operations).

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

Specific objectives:
- 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).
- 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 trees, using the operations of the corresponding data type.

3.- Methodical recursive programming	Learning time: 39h 30m Theory classes: 4h Practical classes: 6h Laboratory classes: 4h Guided activities: 1h Self study : 24h 30m

Description:
Justification of the correctness of recursive algorithms. Function immersion (or generalization). Generalisation of a function. Specification immersions by weakening the postcondition and strengthening the precondition. Relationship between tail-recursive algorithms and iterative algorithms.

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

Specific objectives:
- To describe the main steps in the design of recursive algorithms. To justify the correctness of relatively simple recursive algorithms.
- To know what a generalization of a function is, and to be able to explain the difference between specification gneralisations and efficiency generalisations. To know the different types of specification generalizations and the different types of efiiciency generalisations.
- Given a simple recursive algorithm, to determine whether there is a straightforward way to obtain an equivalent iterative algorithm, and if so, write it.

4.- Methodical iterative programming	Learning time: 24h 30m Theory classes: 2h Practical classes: 3h 30m Laboratory classes: 3h Guided activities: 1h Self study : 15h

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

Related activities:
First Test.

Specific objectives:
- To describe the main steps in the design of iterative algorithms. To justify the correctness of relatively simple iterative algorithms.

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:
Efficiency enhancements for recursive and iterative programs
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.

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).
- 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).
- 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.
- To determine if the efficiency of a given simple iterative algorithm can be improved and, if it is possible, to design a more efficient iterative algorithm that solves the same problem.

340371 - PRO1-I2O23 - Programming I

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 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.

Related activities:
Final Test.

Specific objectives:
- To implement a data structure with specific requirements for its operations and/or the efficiency of such operations, using recursive data types (or linked data structures).
- To design iterative and recursive algorithms for solving search and traversal problems in linked data structures by using directly their representation.

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.

340371 - PRO1-I2O23 - Programming I

Bibliography

Basic:

Musser, David R.; Derge, Gillmer J.; Saini, Atul. STL tutorial and reference guide : C++ programming with the standard template library. 2nd ed. Boston [etc.]: Addison-Wesley publishing company, 2000. ISBN 9780321702128.

Weiss, Mark Allen. Data structures and problem solving using C++. 2a ed. Upper Saddle River: Pearson Education International, 2003. ISBN 0321205006.

Complementary:

Balcázar, José Luis. Programación metódica. Madrid [etc.]: McGraw-Hill, 1993. ISBN 8448119576.

Others resources:

Hyperlink

http://c.conclase.net/curso/index

C++ course. To be used as a reference manual.

http://www.cplusplus.com/reference/stl/

Reference manual for C++ STL containers.