2010-11 academic year

Programming Fundamentals (21406)

Degree/study: Bachelor's degree in Computer Sciences, Bachelor's degree in Telematics Engineering and Bachelor's degree in Audiovisual Systems Engineering
Year: 1st
Term: 2nd and 3rd
Number of ECTS credits: 8 credits
Time commitment: 200 hours
Teaching language or languages: catalan and spanish
Teaching Staff: Jesús Ibañez

1. Presentation of the subject

Programming Fundamentals is part of a set of subjects about algorithmics and programming carried out in the first and second year of the bachelor's degrees in Computer Sciences, Telematics Engineering and Audiovisual Systems Engineering. For this first subject of the set, it is assumed that students have no prior knowledge about algorithmics and programming. In this course the foundations of algorithmics, of the data structures and of the programming in C language are also established. During the subjects called Object-Oriented Programming and Structure of Data and Algorithms (first term of the second year) the skills acquired here will be studied deeply, so at the end of the first term of the second year students will have to be able to develop programs with a considerable size using appropriate data structures, in an imperative way and in an object-oriented way. It is also important to note that the fundamentals acquired in Programming Fundamentals are essential to implement the practical part of many subjects of the degree.

The learning activities are mainly divided into different categories depending on their type:
- Lectures: teachers explain a series of concepts and techniques, and also examples of its use. Students must revise teacher's explanations and their own notes out of the classroom to assimilate the contents.
- Seminar sessions: students must solve a series of small activities, putting into practice the concepts and techniques explained in the lectures. These activities will begin in computer classrooms and will have to be finished out of the classroom.
- Practical sessions: students have to solve some larger problems than the exercises mentioned above, so they must decide which concepts and techniques they have to use in each case. These activities will begin in computer classrooms and they have to be finished out of the classroom. Moreover, in practical sessions students learn how to use programming tools.
- Self-assessment exercises: exercises to help students to check if they have assimilated the concepts and techniques presented at the end of each unit. This activity is carried out by students out of the classroom.

2. Previous requirements to follow the formative itinerary

This course doesn't require previous knowledge on programming or algorithmics. To carry out some exercises it is required some knowledge on mathematics learnt in high school.

3. Competences to be obtained in the subject

The main objective of this subject is that students acquire the fundamentals of algorithmics and data structures, as well as they learn to create fluently medium size programs using C language.
This chapter details what students are expected to have learned once the subject is finished.

First, the general competences refer to skills not directly related to programming itself, but to a professional context of an engineer. The specific competences refer to aspects of the subject.

3.1. General competences

Instrumental
- CG1: Capacity to synthesise
Students should be able to write solutions with the essential elements, in a simple and elegant way and as efficiently as possible.
- CG2: Capacity of analysis
Students should be able, based on a specific problem, to analyze it and propose appropriate solutions.

Systemic
- CG3: Capacity to put knowledge into practice
Students should be able to apply knowledge acquired to solve specific problems, choosing the technique that best suits each case.
- CG4: Interest in quality
Students should be able to have a code that is, apart from efficient, easy to read and maintain. It is also important to properly document both in the same code and in a report.

3.2. Specific competences

- CE1: Capacity to work with programming tools
Students should be able to work with the basic programming tools: a compiler and a debugger. They must be able to work with a programming editor and an IDE. This competence is essential for a proper development of the others.
- CE2: Control the basic and complex static data types
Students should be able to distinguish the different types of basic and complex static data and decide the appropriate type in each specific situation.
- CE3: Control control structures
Students should be able to distinguish the different complex control structures and decide the most appropriate to solve specific problems.
- CE4: Capacity to solve problems through descendant design and the control of the use of functions and libraries.
Students should be able to solve problems of more complexity using descendant design techniques. Specially, students must understand the operation of function calls and parameters pass and they must control the use and creation of libraries, and they must be able to divide a problem into the appropriate units.
- CE5: Control of dynamic data types and dynamic memory management
Students must understand the mechanism of memory management, as well as the use of pointers and dynamic control of data structures. It is also included the treatment of text files.
- CE6: Documentation and code structure
Students must acquire the habit of structuring and documenting code properly in order to facilitate further readings.
- CE7: Capacity to read (quickly) code in C
Students should be able to understand code written by other programmers, quickly.
- CE8: Control of algorithmics fundamental elements
Students should know and be able to apply properly the fundamental concepts of algorithms, such as recursion and searching and sorting algorithms.
- CE9: Capacity of algorithm analysis
Students should know the notation and the mechanisms of the analysis of algorithmic complexity, and must be able to calculate the run time of a program.

4. Contents

Unit 1: Introduction and general concepts

Concepts	Procedures
- Short history about programming and its languages and paradigms - Compilation and interpretation mechanisms - Difference between program and algorithm	

Unit 2: Basic data types

Concepts	Procedures
- Variables and constants - Basic data types: - Numeric types - Characters - Booleans	- Declaration of constants and variables of different types

Unit 3: Expressions, sentences and control structures

Concepts	Procedures
- Expression creation - Introduction to Boolean logic - Sentences or instructions: - Assignations - Input/output operations - Operators precedence order - Control structures: - Conditional structures - Iterative structures	- Expression evaluation - Resolution of small problems using appropriate sentences and control structures

Unit 4: The functional decomposition and the descendant design

Concepts	Procedures
- Descendant design - Declaration of functions - Parameter definition - Function calls and parameters pass - Void type	- Division of problems into subproblems - Resolution of small problems defining the functions properly

Unit 5: Complex static data types

Concepts	Procedures

- One-dimension arrays - Multi-dimension arrays - Strings or character strings - Structures (structs)	- Resolution of small problems about typical operations with each of the data types

Unit 6: Declaration of own types

Concepts	Procedures
- Structure types - Type definition by typedef - Type conversions	- Resolution of small definition problems of own data types

Unit 7: Pointers and dynamic memory management

Concepts	Procedures
- Pointers declaration - Operations of direction and indirection - Value assignation to pointers - Null pointer - Reference parameters pass - Arrays and pointers in C	- Resolution of small problems about typical operations with pointers - Resolution of small typical problems of arrays using pointers

Unit 8: The functional decomposition and the descendant design (part II)

Concepts	Procedures
- Reference parameters pass - Main function and its arguments - Visibility and scope - The libraries and the processor	- Resolution of problems by decomposition using functions with reference parameters

Unit 9: Text files

Concepts	Procedures
- The file type (FILE) - Operations with files: - Open - Close - Write - Read - Standard input/output	- Resolution of small problems about typical operations with text files

Unit 10: Style and good practices

Concepts	Procedures
- Style, legibility and obfuscation - Good practices - Common errors - Code analysis - Depuration	- Resolution of problems of error detection and style improvement. - Use of code analysis and depuration tools.

Unit 11: Functional decomposition and descendant design (part III)

Concepts	Procedures
- Library programming - Segmentation of the code into files - Header files	- Resolution of problems of creation and use of libraries

Unit 12: Search and sorting algorithms

Concepts	Procedures
- Lineal and binary search - Basic sorting algorithms - Bubble - Insertion - Selection	- Resolution of problems about search and sorting algorithms

Unit 13: Recursivity

Concepts	Procedures
- Recursivity concept - Recursive algorithms - Base case and recurrence - Pros and cons - Direct and indirect recursivity - Transformation into iterative algorithms - Specific examples of recursive algorithms (fractals, Hanoi towers, Quicksort, etc)	- Resolution of problems by the definition of recursive functions

Unit 14: Algorithm analysis

Concepts	Procedures

- Program run time - Asymptotic notation - Run time calculation	- Resolution of calculation problems and comparison of programs run time

5. Learning methodology

The normal learning process of a unit begins with a lecture where some theoretical and practical fundamentals are presented. This activity takes place in a large group of students. Students must then complete this activity with a careful reading of the notes. For example, a typical lecture that lasts 2 hours, properly used, will need an autonomous 1-hour work out of the classroom.
Then one or some seminar sessions or practical sessions take place. In the seminar sessions students put into practice the concepts and techniques presented in the lectures, by implementing programs to solve small problems. The aim is that students consolidate the fundamentals to later carry out more complex problems. This activity must be done individually, in a group of about 15 students. Each activity of this kind is planed to last about 4 hours, 2 of which are done with the help of the teacher. The first activities of the session, with solutions offered by the teacher, must be solved before going into the classroom. The teacher can request a hand-in of one of the activities proposed at the end of each session.
In the practical sessions some more complex problems are proposed, especially in the 3 evaluable practical activities, which require a preliminary design of the solution that will be implemented and which integrates different concepts and techniques. In the final practical exercise all the specific skills that students must acquire in this subject are joined. Each of these activities of this type is performed by couples, in a group of about 30 students, and they must continue out of the classroom.
In each practical and seminar session, some time is devoted to discuss the main problems presented in the previous session.
The last step of the unit is to solve self-assessment exercises by which students can check that they have acquired the skills evaluated later in the partial and final exams.

5.1. Learning units

Unit 1: First steps: introduction, basic data types and control structures

Units	Learning activities			
Unit 1: Introduction and general concepts Unit 2: Basic data types Unit 3: Expressions, sentences and control structures	Lectures	Seminar sessions	Practical sessions	Self-assessment
	T1, T2 and T3	S1	P1	A1

Time commitment of unit 1: 17 hours (10 in the classroom, 7 out of the classroom)

Details about activities:
- Lecture T1. 3 hours (2 in the classroom, 1 out of the classroom): short history of programming and explanation of compilation and interpreting models.
- Lectures T2 and T3. 6 hours (each session: 2 in the classroom, 1 out of the classroom): explanation of basic data types and main examples of use. Explanation of basic sentences and expressions creation with the main examples of its use. Conditional and iterative control structures and main examples of its use.
- Seminar session S1. 4 hours (2 in the classroom, 2 out of the classroom): activities on

expressions and conditional and iterative control structures creation.
- Practical session P1. 3 hours (2 in the classroom, 1 out of the classroom): it will be explained how to install a compiler, and how to edit, compile and run a program. The students will be given small programs with errors because they get used of the typical error messages of the compiler.
- Self-assessment A1. 1 hour (out of the classroom): resolution of self-assessment activities about expressions and control structures creation.

Unit 2: Functions and descendant design

Units	Learning activities			
Unit 4: The functional decomposition and the descendant design	Lectures	Seminar sessions	Practical sessions	Self-assessment
	T4	S2		A2

Time commitment unit 2: 8 hours (4 in the classroom, 4 out of the classroom)

Details about activities:
- Lecture T4. 3 hours (2 in the classroom, 1 out of the classroom): explanation on descendant design fundamentals, as well as the function definition and the call mechanism and value parameters pass, with examples of its use.
- Seminar session S2. 4 hours (2 in the classroom, 2 out of the classroom): activities about descendant design and value parameters pass.
- Self-assessment A2. 1 hour (out of the classroom): Resolution of self-assessment activities about functions and value parameters pass.

Unit 3: Complex static dada types and type declaration

Units	Learning activities			
Unit 5: Complex static data types	Lectures	Seminar sessions	Practical sessions	Self-assessment
Unit 6: Declaration of own types	T5 and T6	S3	P2 and P3	A3

Time commitment unit 3: 29 hours (10 in the classroom, 19 out of the classroom)

Details about activities:
- Lectures T5 and T6. 6 hours (each session 2 in the classroom, 1 out of the classroom): explanation of complex data types (arrays, strings and structures) with examples of use. Explanation of the definition of own data types by the programmer, with examples of use.
- Seminar session S3. 4 hours (2 in the classroom, 2 out of the classroom): activities about complex data types.
- Practical sessions P2 and P3 (first evaluated practical activity). 18 hours (each, 2 in the classroom, 7 out of the classroom): students must solve (programming in C) medium-size problems, using arrays, strings and structures and, in some cases, defining the own types. It includes the creation of an explicative work report.
- Self-assessment A3. 1 hour (out of the classroom): Resolution of self-assessment activities on complex data types.

Unit 4: Pointers and parameters pass

Units	Learning activities

Unit 7: Pointers and dynamic memory management Unit 8: The functional decomposition and the descendant design (part II)	Lectures	Seminar sessions	Practical sessions	Self-assessment
	T7 and T8	S4	P4	A4

Time commitment unit 4: 18 hours (8 in the classroom, 10 out of the classroom)

Details of activities:
- Lectures T7 and T8. 8 hours (each session 2 in the classroom, 2 out of the classroom): explanation on the operation of pointers with examples of use. Explanation on reference parameters pass with examples of use. Explanation on the visibility norms.
- Seminar session S4. 4 hours (2 in the classroom, 2 out of the classroom): activities on pointers and reference parameters pass.
- Practical session P4. 5 hours (2 in the classroom, 3 out of the classroom): students must solve (programming in C) some small problems, using pointers and reference parameters pass.
- Self-assessment A4. 1 hour (out of the classroom): Resolution of self-assessment activities about pointers and reference parameters pass.

Unit 5: Text files

Units	Learning activities			
Unit 9: Text files	Lectures	Seminar sessions	Practical sessions	Self-assessment
	T9	S5		A5

Time commitment unit 5: 8 hours (4 in the classroom, 4 out of the classroom)

Details of activities:
- Lecture T9. 3 hours (2 in the classroom, 1 out of the classroom): explanation on text files and examples of use.
- Seminar session S5. 4 hours (2 in the classroom, 2 out of the classroom): Activities on text files.
- Self-assessment A5. 1 hour (out of the classroom): Resolution of self-assessment activities on files.

Unit 6: Good practices and library programming

Units	Learning activities			
Unit 10: Style and good practices Unit 11: Functional decomposition and descendant design (part III)	Lectures	Seminar sessions	Practical sessions	Self-assessment
	T10 and T11		P5, P6, P7 and P8	A6

Time commitment unit 6: 33 hours (12 in the classroom, 21 out of the classroom)

Details of activities:
- Lecture T10. 3 hours (2 in the classroom, 1 out of the classroom): explanation on good practices to program in readable way and obtain programs with fewer errors. Explanation on common errors while codifying a program. Explanation on analysis techniques and code

depuration.
- Lecture T11. 3 hours (2 in the classroom, 1 out of the classroom): explanation on the process of library creation and code segmentation in some files. Review of visibility norms.
- Practical session P5. 4 hours (2 in the classroom, 2 out of the classroom): students will start to use an IDE and it will be explained how to debug a program using the IDE. Students will debug some programs.
- Practical session P6. 4 hours (2 in the classroom, 2 out of the classroom): it will be explained how to create a complex program of several modules (or files) using an IDE. Students will create a program divided into several modules.
- Practical sessions P7 and P8 (second evaluated practical activity). 18 hours (each session, 2 in the classroom, 7 out of the classroom): students must solve (programming in C) complex problems that integrate pointers, reference parameters pass, text files and segmentation of code into several files. The resolution must be programmed according the recommendations of good practices. It includes the creation of an explicative work report.
- Self-assessment A6. 1 hour (out of the classroom): Resolution of self-assessment activities about style, good practices, and segmentation of code into modules or files.

Unit 7: Search, sorting and recursivity

Units	Learning activities			
Unit 12: Search and sorting algorithms Unit 13: Recursivity	Lectures	Seminar sessions	Practical sessions	Self-assessment
	T12, T13, T14 and T15	S6 and S7		A7

Time commitment unit 7: 24 hours (12 in the classroom, 12 out of the classroom)

Details of activities:
- Lecture T12. 3 hours (2 in the classroom, 1 out of the classroom): explanation on binary and lineal search algorithms, as well as basic sorting algorithms (bubble, insertion and selection).
- Lectures T13, T14 and T15. 12 hours (T13, 2 in the classroom, 2 out of the classroom; T14, 2 in the classroom, 3 out of the classroom; T15, 2 in the classroom and 1 out of the classroom): explanation on the recursivity concept. Explanation of the fundamentals of recursive algorithms and its pros and cons. Explanation of the different types of recursivity. Explanation of the transformation of recursive algorithms into iterative algorithms. Discussion on some specific examples of algorithms.
- Seminar session S6. 4 hours (2 in the classroom, 2 out of the classroom): Activities on search algorithms, sorting algorithms and recursive algorithms.
- Seminar session S7. 4 hours (2 in the classroom, 2 out of the classroom): Activities on recursive algorithms.
- Self-assessment A7. 1 hour (out of the classroom): Resolution of self-assessment activities on search algorithms, sorting algorithms and recursivity.

Unit 8: Algorithm analysis

Units	Learning activities			
Unit 14: Algorithm analysis	Lectures	Seminar sessions	Practical sessions	Self-assessment
	T16, T17 and T18	S8	P9 and P10	A7

Time commitment unit 8: 33 hours (12 in the classroom, 21 out of the classroom)

Details of activities:
- Lectures T16 and T17. 7 hours (T16, 2 in the classroom, 2 out of the classroom; T17, 2 in the classroom, 1 out of the classroom): explanation on implied factors of the efficiency of an algorithm. Explanation of the asymptotic notation. Explanation of the procedures to calculate the run time of an algorithm, with illustrating examples.
- Lecture T18. 3 hours (2 in the classroom, 1 out of the classroom): general review and doubt resolution.
- Seminar session S8. 4 hours (2 in the classroom, 2 out of the classroom): Activities on the analysis of the complexity of algorithms.
- Practical sessions P9 and P10 (third evaluated practical activity). 18 hours (each session, 2 in the classroom, 7 out of the classroom): students have to solve (programming in C) a complex problem that integrates all the concepts and techniques of the subject (including recursive algorithms and algorithmic complexity analysis). The solution must be programmed according to the recommendations of good practices. It includes the creation of an explicative work report.
- Self-assessment A8. 1 hour (out of the classroom): Resolution of self-assessment activities about algorithm complexity analysis.
- Important: apart from the planned working hours which are described above, students must dedicate 10 hours (8.5+1.5) to prepare and take the partial examination, and 20 (17.5+2.5) hours for the final exam.
- Total time commitment: 200 hours (76 in the classroom, 124 out of the classroom)

6. Avaluation

6.1. General evaluation criteria

This course consists of two main parts:
- Theoretical and practical fundamentals, which include activities related to the lectures and seminars
- Practical activities

Each of these parts represents 50% of the final mark, but the students have to pass both parts to pass the subject.

Theoretical and practical fundamentals
- There will be two examinations, one in the middle of the subject (at the end of the second term) and another one at the end of the subject (at the end of the third term). The first one means a 25% of the mark and it is focused on the activities done during second term practical sessions. The second exam is 75% of the mark and covers practical and theoretical topics of the whole subject.

- In the seminar sessions, teachers will review students' work during all the term, and they can ask for a hand-in of an activity at the end of each session. Based on these reviews and hands-in, the mark of the theoretical and practical fundamentals can increase (but never decrease) at maximum 1,5 points.

- Furthermore, some self-assessment activities are planned, one at the end of each unit, so that students can assess their progress. These activities consist of a series of multiple-choice questions, similar to the exams. Also, in seminar activities, students can also evaluate their progress by comparing their solutions with the results given. In any case, these self-assessment activities do not affect the mark of the subject.

Practical activities
- The mark of this part is obtained from three activities to hand in, distributed throughout the subject. Each of these practices has the same value in the mark (i.e., each evaluable practical activity represents 33.3%). Regarding the practical activities mark, apart from the overall assessment, teachers will be able to review the progress in the work of students throughout the practical sessions. If a group does not do this review, or if this review is not satisfactory, its students must defend the practical activity after their hand-in.

- Apart from the activities to hand-in, other practical activities will be useful for students to evaluate their progress. These activities do not affect the mark.

September examination sitting
- In the case that students do not pass both parts in the June sitting, it will only be kept one part passed in case that the mark of the other part is equal or greater than 3. Otherwise, students must retake the examination and they have to hand in a new practical activity (with different instructions).

6.2. Concretion

First evaluated practical activity
- It will be evaluated the resolution of a practical activity in which students can show the degree of assimilation of concepts and skills acquired during the first third of the course. In particular, the skills assessed will be CE2, CE3 and certain aspects of CE4 and CE6 (code structure discussed above). The practical activity consists of solving a problem, including its analysis and programming of an appropriate solution in C language. Students must choose static data structures as well as the operations flow of the solution and perform a simple descendant design. It will also be assessed the four general competences defined.

Second evaluated practical activity
- It will be assessed the resolution of a practice in which students can show the degree of assimilation of concepts and skills acquired during the first two thirds of the course, especially during the second. In particular, the skills assessed will be CE2, CE3, CE4, CE5 and CE6. The practical activity consists of solving a complex problem, including its analysis, the choice of appropriate data structures (static and dynamic ones), the descending design, the planning of an adequate solution in C (following recommendations of good practices) and their proper documentation in a report where it will also be justified the decisions taken. It will also be assessed the four general competences defined.

Third evaluated practical activity
- It will be assessed the resolution of a practical activity in which students can show the degree of assimilation of concepts and skills acquired throughout the subject. In particular, the skills assessed will be CE2, CE3, CE4, CE5, CE6, CE8 and CE9. The practical activity consists of solving a complex problem, including its analysis, the choice of appropriate data structures (static and dynamic ones), the descendant design, the planning of an appropriate solution in C (following the recommendations of good practices) and their proper documentation in a report where it also be justified the decisions taken and which will include a part of analysis of the algorithmic complexity. The four general competences defined will also be assessed.

Partial examination
- It will be evaluated the understanding and application of concepts and techniques acquired during the first half of the subject (second term). In particular, the skills assessed will be CE2, CE3, CE4, CE5 and especially CE7. The evaluation method involves an examination of approximately 20 multiple-choice questions with four options for each question with only one correct answer. Questions will be taken (with small modifications) from the proposals for practical sessions and self-assessment activities. The evaluation will take place during the examination period of the second term.

Final examination
- It will be evaluated the understanding and application of concepts learnt throughout the whole course in small programs. In particular, the skills assessed will be CE2, CE3, CE4, CE5, CE7, CE8 and CE9. The evaluation method involves an examination of approximately 30 multiple-choice questions with four options for each question with only one correct answer. The evaluation will take place during the examination period of the third term.

7. Bibliography and teaching resources

7.1. Basic bibliography

- Toni Navarrete Terrasa. Introducción a la programación con lenguaje C.
- Jesús Bisbal Riera. Manual d'algorísmica: Recursivitat, complexitat i disseny d'algorismes. Editorial UOC. ISBN: 978-84-9788-570-6
- Brian W. Kernighan, Dennis M. Ritchie: El lenguaje de programación C. Segunda edición. Prentice-Hall. ISBN: 968-880-205-0

7.2 Learning information resources. Additional bibliography.

Other books about C:
- Herbert Schildt: C Manual de referencia. Mc Graw Hill. 84-481-0335-1
- James L. Antonakos, Kenneth C. Mansfield Jr.: Programación estructurada en C. Prentice-Hall. ISBN: 84-89660-23-9
- Marco A. Peña, José M. Cela: Introducción a la programación en C. Edicions UPF. ISBN: 84-8301-429-7
- Luis Joyanes, Ignacio Zahonero: Programación en C. Mc Graw Hill. ISBN: 84-481-3013-8
- Félix García, Jesús Carretero, Javier Fernández, Alejandro Calderón: El lenguaje de programación C. Diseño e implementación de programas. Prentice-Hall. ISBN: 84-205-3178-2
- P.J. Plauger, Jim Brodie: C Estándar. Guía de referencia rápida para programadores. Anaya. ISBN: 84-7614-264-1

Other "classical" books on algorithmics (more complex):
- Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft: Estructuras de datos y algoritmos. Addison Wesley, 1988. ISBN: 968-444-345-5
- Niklaus Wirth: Algoritmos + estructuras de datos = programas. Ediciones del Castillo, 1980. ISBN: 84-219-0172-9
- Niklaus Wirth: Algoritmia y estructuras de datos. Prentice Hall, 1987. ISBN: 968-880-113-5
- D.E. Knuth: El arte de programar ordenadores (3 volums). Editoria Reverté. ISBN: 84-291-2661-9
- Terrence W. Pratt, Marvin V. Zelkowitz: Lenguajes de programación. Diseño e implementación. Prentice Hall, 3ª edic., 1997. ISBN: 970-17-0046-5
- G. Brassard, P. Bratley: Fundamentos de algoritmia. Prentice-Hall. ISBN: 84-89660-00-X

7.3 Didactic resources. Subject learning material.

In Aula Moodle of the subject (Aula Global) it will be uploaded the learning material of the subject. In particular:
- Notes
- Instructions for the seminar sessions
- Instructions for the practical sessions
- Self-assessment activities