200011 - INF - Computer Science

-Coordinating unit: 200 - FME - School of Mathematics and Statistics
-Teaching unit: 723 - CS - Department of Computer Science
749 - MAT - Department of Mathematics

-Academic year: 2018
-Degree: BACHELOR'S DEGREE IN MATHEMATICS (Syllabus 2009). (Teaching unit Compulsory)
-ECTS credits: 7.5

Teaching staff

Coordinator: JORDI CORTADELLA FORTUNY

Others: Primer quadrimestre:
  GUILLEM BLANCO FERNÁNDEZ - A, B
  M. JOSE BLESA AGUILERA - A, B
  JORDI CORTADELLA FORTUNY - A, B
  ÓSCAR RIVERO SALGADO - A, B
  SALVADOR ROURA FERRET - A, B

Prior skills

Capability for abstract reasoning.

Requirements

Knowledge of basic informatics tools at user level.

Degree competences to which the subject contributes

Specific:
  1. CE-2. Solve problems in Mathematics, through basic calculation skills, taking in account tools availability and the constraints of time and resources.
  3. CE-4. Have the ability to use computational tools as an aid to mathematical processes.

Generical:
  4. CB-1. Demonstrate knowledge and understanding in Mathematics that is founded upon and extends that typically associated with Bachelor's level, and that provides a basis for originality in developing and applying ideas, often within a research context.
  5. CB-2. Know how to apply their mathematical knowledge and understanding, and problem solving abilities in new or unfamiliar environments within broader or multidisciplinary contexts related to Mathematics.
  6. CB-3. Have the ability to integrate knowledge and handle complexity, and formulate judgements with incomplete or limited information, but that include reflecting on social and ethical responsibilities linked to the application of their knowledge and judgements.
  7. CG-1. Show knowledge and proficiency in the use of mathematical language.
  9. CG-3. Have the ability to define new mathematical objects in terms of others already know and ability to use these objects in different contexts.
  10. CG-4. Translate into mathematical terms problems stated in non-mathematical language, and take advantage of this translation to solve them.
  12. CG-6 Detect deficiencies in their own knowledge and pass them through critical reflection and choice of the best action to extend this knowledge.

Transversal:
2. SELF-DIRECTION. Detecting gaps in one's knowledge and overcoming them through critical self-appraisal. Choosing the best path for broadening one's knowledge.

Teaching methodology

The theory classes present the basic theoretical basis necessary for building programs.

Problems sessions are designed for solving exercises with pencil and paper, to consolidate the acquired knowledge and to design algorithms for solving the statements posed. They are intended as a series of participatory sessions in which the student engages with ideas and present their solutions. Prior preparation by the student is required.

In the lab sessions, the student performs individually, with the help of teachers, programming exercises that demonstrate the use of the concepts taught in theory classes.

Over the course we introduce the theoretical components that must be assimilated by students. To this end, we believe that the most convenient method is the resolution of problems that require the tool or concept introduced. It is therefore essential student personnel work in the design and implementation of programs. This effort will be supported by self-learning tools.

As a complement self-learning tools will be provided so that students can consolidate their knowledge of programming during the hours of study outside the classroom. Specifically, students will have access to a version adapted to the contents of the subject of a self-learning tool for programming, the "Jutge" developed within the Department of Languages and Informatic Systems by a team of teachers led by Jordi Petit and Salvador Roura.

Learning objectives of the subject

The overall objective of the course is that the student be able to write fluently and legibly correct programs to solve problems of medium difficulty, based on processing sequences, and basic difficulty in other areas, in particular problems with mathematical formulation. Another aim is to familiarize students with a computing environment and a current programming language, in this case C++. Students must learn, first, to design and implement algorithms and, second, to use other tools such as editors and compilers.

Specific objectives:

- Making students feel comfortable and reliable in the design of programs written in an imperative language.
- Learn the basic algorithms with elementary and structured data (prime numbers, gcd, traversals, searching, sorting, matrices ...).
- To apply the inductive method for solving complex problems.
- To use editing, compilation and execution tools to code and run programs.

Study load

<table>
<thead>
<tr>
<th>Study load</th>
<th>Total learning time: 187h 30m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours large group:</td>
<td>30h 16.00%</td>
</tr>
<tr>
<td>Hours medium group:</td>
<td>0h 0.00%</td>
</tr>
<tr>
<td>Hours small group:</td>
<td>45h 24.00%</td>
</tr>
<tr>
<td>Guided activities:</td>
<td>7h 30m 4.00%</td>
</tr>
<tr>
<td>Self study:</td>
<td>105h 56.00%</td>
</tr>
</tbody>
</table>
# Content

## 1. The structure of a computer. Procedures and instructions.

**Learning time:** 15h  
Theory classes: 2h 30m  
Practical classes: 3h  
Self study: 9h 30m

**Description:**  
Basic orders in Linux. Text editors.

## 2. Variables and basic instructions.

**Learning time:** 31h 30m  
Theory classes: 5h 30m  
Practical classes: 6h  
Self study: 20h

**Description:**  
Completion and correction.  
Basic syntax of the C++. Writing, compiling and running a program in C++.

## 3. Treatment sequences.

**Learning time:** 41h  
Theory classes: 7h  
Practical classes: 10h  
Self study: 24h

**Description:**  
The concept of sequence. Traversing and searching. Examples. Traversing and searching schemes.
### 4. Actions and functions.

**Description:**

- Introduction to recursion.
- Methods and functions in C++. Side effects.

**Learning time:** 29h 30m
- Theory classes: 5h 30m
- Practical classes: 5h
- Self study: 19h

### 5. Data not elementary.

**Description:**

- Down design. Efficiency.
- The vector class. C++ syntax.

**Learning time:** 41h
- Theory classes: 7h
- Practical classes: 10h
- Self study: 24h

### 6. Tuples and classes.

**Description:**
Non-homogeneous data. Basic notions of objects. Examples of use.

- Object-oriented design.

**Learning time:** 28h
- Theory classes: 5h
- Practical classes: 5h
- Self study: 18h
The assessment takes into account the following components:

- Knowledge and use of algorithms and techniques introduced in the course
- Algorithmic problem-resolution.
- Ability to program in C++ simple program.
- Ability to problem-solving mid-level programming.

There will be a programming partial test (PL) which is performed in the laboratory, a final programming test (FL) which is performed in the laboratory, a final written exam (FT) consisting of programming programs and exercises.

The final grade is calculated according to the formula:

\[ 0.6 \times \max \{0.3 \, \text{PL} + 0.7 \, \text{FL}, \, \text{FL}\} + 0.4 \, \text{FT} \]

An extra exam will take place on July for students that failed during the regular semester.

**Qualification system**

The "Jutge" will be used in conducting laboratory tests, partial and final, providing the same software development environment, to aid them during the tests. This tool will also support the development of the programming project. In none of the tests it is allowed the use of books or notes.
Bibliography

Basic:


Complementary:


Others resources:

Hyperlink

Introduction to Programming

http://www.cs.upc.edu/ jordicf/Teaching/FME/Informatica