UNIVERSIDAD AUTÓNOMA DE MADRID

17821 - PROGRAMMING II

Information of the subject

Code - Course title: 17821 - PROGRAMMING II

Degree: 473 - Graduado/a en Ingeniería Informática
474 - Graduado/a en Ingeniería Informática y Matemáticas
722 - Graduado/a en Ingeniería Informática
734 - Graduado/a en Ingeniería Informática y Matemáticas (2019)

Faculty: 350 - Escuela Politécnica Superior

Academic year: 2020/21

1. Course details

1.1. Content area

Programación y estructuras de datos

1.2. Course nature

Compulsory

1.3. Course level

Grado (EQF/MECU 6)

1.4. Year of study

1

1.5. Semester

Second semester

1.6. ECTS Credit allotment

6.0

1.7. Language of instruction

Español

1.9. Recommendations

No academic prerequisites are needed to attend this course, but basic knowledge on programming fundamentals and software design, using the C programming language, is assumed. It is recommended to have completed the courses Programming I and Practical Computer Workshop.

In particular, before taking this course the student should be able to:

- Design algorithms using pseudocode.
- Understand and implement recursive algorithms.
- Translate pseudocode into C code.
- Use fluidly an Integrated Development Environment (IDE) to edit, compile, link and debug code.
- Apply basic techniques of cooperative project management.

1.10. Minimum attendance requirement

CONTINUOUS ASSESSMENT METHOD WITH MANDATORY CLASS ATTENDANCE

Attendance to at least 85% of the lectures is mandatory.

NON-CONTINUOUS ASSESSMENT METHOD WITHOUT MANDATORY CLASS ATTENDANCE

Attendance to the lectures is recommended, but not mandatory.

1.11. Subject coordinator/s

Rosa Maria Carro Salas

https://autoservicio.uam.es/paginas-blancas/

1.12. Competences and learning outcomes

1.12.1. Competences

C14 Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.
B4 Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.
C3 Capacidad para comprender la importancia de la negociación, los hábitos de trabajo efectivos, el liderazgo y las habilidades de comunicación en todos los entornos de desarrollo de software.
C4 Capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.
C5 Conocimiento, administración y mantenimiento sistemas, servicios y aplicaciones informáticas.
C6 Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.
C7 Conocimiento, diseño y utilización de forma eficiente los tipos y estructuras de datos más adecuados a la resolución de un problema.
C14 Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.

1.12.2. Learning outcomes

- Understanding of programming techniques using a high level language.
- Understanding of abstract data types, as well as their implementation using a high level programming language.
- Understanding the evolution of data abstraction: from abstract data types to object orientation.
- Good use of programming.

1.12.3. Course objectives

General objectives:

- Appreciate the benefits of data abstraction in the development of algorithms and software applications.
- Distinguish between an abstract data type and its implementation.
- Identify, design and effectively implement the most appropriate abstract data types for a given problem.
- Apply modularization as a conceptual tool to develop programs and software applications.
- Participate in the analyses and group discussions that are established in the course of the development of the proposed activities, cooperate with other students in the development of joint work and communicate properly and correctly their reflections on the results of their work.

Unit by unit specific objectives:

1. **Abstract Data Types**

1.1 Explain the process of data abstraction.

1.2 Distinguish between primitive types and abstract data types.

1.3 Enumerate the benefits of data abstraction in the development of algorithms and software applications.

1.4 Identify the abstract data types that are best suited for a given problem.

1.5 Apply abstract data types to algorithm design.

1.6 Distinguish between an abstract data type and its implementation.

1.7 Evaluate the benefits of the different implementations of an abstract data type in terms of space and time requierements

1.8 Implement different abstract data types in the C programming language

2. **Stacks and applications**

2.1 Describe the organization and data access in a stack.

2.2 Describe examples of stack application.

2.3 Provide an informal specification of the stack ADT.

2.4 Distinguish between the stack ADT and its different implementations.

2.5 Evaluate the benefits and weaknesses of different stack implementations.

2.6 Implement in the C language the stack primitives with different data structures.

2.7 Evaluate an arithmetic expression in infix, prefix or postfix notation.

2.8 Express pseudocode of different algorithms for the evaluation and translation of arithmetic expressions.

2.9 Implement in the C language different algorithms for the evaluation and translation of arithmetic expressions.

3. **Queues and applications**

3.1 Describe the organization and data access in a queue.

3.2 Identify the fundamental differences between a queue and a stack.

3.3 Provide an informal specification of the queue ADT.

3.4 Apply the queue ADT to solve paradigmatic problems.

3.5 Evaluate the benefits and weaknesses of different queue implementations.

3.6 Implement in the C language the queue primitives with different data structures.

4. **Lists and linked lists**

4.1 Distinguish between sequential and linked structures

4.2 Provide an informal specification of the simple linked list ADT.

4.3 Evaluate the benefits and weaknesses of different linked list implementations.

4.4 Implement in the C language the linked list primitives using arrays.

4.5 Implement in the C language the linked list primitives using dynamic memory.

4.6 Evaluate the benefits and weaknesses of linked lists for the implementation of other ADT (such as sets, queues or stacks)

4.7 Implement queues and stacks using linked lists.

5. **Trees. Binary trees and sorted trees**

5.1 Describe the characteristics of hierarchical structures.

5.2 Enumerate the different tree types.

5.3 Describe the different algorithms for binary tree traversal.

5.4 Provide an informal specification of the binary tree ADT.

5.5 Implement in the C language the binary tree primitives using linked nodes.

5.6 Define the binary search tree BST.

5.7 Evaluate the time complexity of BST operations.

5.8 Use BSTs to implement different ADTs.

5.9 Build the binary expression tree for an arithmetic expression.

6. **Priority queues and heaps**

6.1 Provide an informal specification of the priority queue ADT.

6.2 Implement priority queues using linked lists.

6.3 Describe the binary heap data type.

6.4 Implement in the C language the binary heap using arrays.

6.5 Implement the basic heap operations with logarithmic complexity.

6.6 Sort a list of elements using the heap-sort algorithm.

7. **Recursion**

7.1 Describe the execution stack of a function that calls itself.

7.2 Design recursive algorithms.

7.3 Demonstrate the efficiency of a recursive algorithm using the induction method.

7.4 Apply the divide and conquer technique for solving programming problems.

7.5 Compare iterative and recursive solutions to a given problem.

7.6 Eliminate recursive calls using a stack.

1.13. Course contents

1. Abstract data types

1.1. Primitive data types

1.2. Abstract data types (ADTs)

1.3. Implementation of ADTs

1.4. Modularity

1.5. Object orientation

2. Stacks and applications

2.1. Introduction and first examples

2.2. Stacks

2.3. Stack implementation

2.4. Arithmetic expressions

3. Queues and applications

3.1. Introduction and first examples

3.2. Queues

3.3. Queue implementation

3.4. Queue applications

4. Lists

4.1. Introduction and first examples

4.2. Linked lists

4.3. Linked list implementation

4.4. Other list structures

4.5. Use of linked lists to implement queues and stacks

5. Trees. Binary trees and sorted trees

5.1. Trees

5.2. Binary trees

5.3. Tree traversal

5.4. Binary tree implementation

5.5. Expression trees

5.6. Binary search trees

6. Priority queues and heaps

6.1. Priority queues

6.2. Priority queue implementation

6.3. Heaps

6.4. Heap-sort

7. Recursion

7.1. Recursive definitions and processes

7.2. Recursion in C

7.3. Recursive programs

7.4. Complexity of recursive programs

7.5. Simulation of recursive algorithms

1.14. Course bibliography

Basic:

- Data Structures Using C and C++, Langsam, Augenstein, Tenenbaum. Prentice Hall, 1996. INF/C6120/LAN

Complementary (data structures and abstract data types):

- Introduction to algorithms, Cormen, Leiserson, Rivest, The MIT Press 2001. INF/510.5/COR
- Data structures and algorithms, Aho, Hopcroft, Ullman, Addison-Wesley 1998. INF/C6120/AHO
- Algoritmos y estructuras de datos: una perspectiva en C, Joyanes Aguilar L., Zahonero Martínez I., Mc Graw Hill 2003. INF/510.5/JOY
- Data Structures and algorithm analysis in C, Weiss Mark Allen, Addison Wesley-Longman 1998 INF/681.3.01/WEI

Complementary (C programming):

- The C programming language. Brian W. Kernighan, Dennis M. Ritchie. Prentice Hall, 1988. INF/681.3.062-C/KER
- A book on C. Programming in C. (fourth edition). Kelley, Al and Pohl, Ira. Addison-Wesley, 2004. INF/681.3.062-C/KEL

Complementary (methodology of programming):

- The practice of programming. B.W. Kernighan, R. Pike. Addison-Wesley, 1999. INF/C6110/KER.

2. Teaching-and-learning methodologies and student workload

2.1. Contact hours

	#horas
Contact hours (minimum 33%)	79
Independent study time	71

2.2. List of training activities

Activity	# hours
Lectures	42
Seminars	
Practical sessions	
Clinical sessions	
Computer lab	26
Laboratory	
Work placement	
Supervised study	
Tutorials	
Assessment activities	11
Other	71

3. Evaluation procedures and weight of components in the final grade

3.1. Regular assessment

This course involves lectures, problem classes, lab assignments, two midterm tests and a final exam.

This course is evaluated using the following scheme:

Final grade = 0.4*LAB + 0.6*TH

where LAB is the grade obtained in the lab assignments and TH is the grade for the theory part of the course. Both parts are evaluated over 10 points.

It is necessary to have a pass grade (≥ 5) in both parts to pass the course. Otherwise the final grade will be computed as:

Final grade = min (4.9, 0.4* LAB + 0.6*TH)

The grade for the theory part (TH) can be obtained in two different ways:

CONTINUOUS ASSESSMENT METHOD WITH MANDATORY CLASS ATTENDANCE

- Attendance to at least 85% of the lectures is mandatory.
- First midterm test (PL1): It is necessary to have a pass grade (PL1 ≥ 5) to pass this test.
- Second midterm test (PL2): Only students that have passed the first midterm test can take the second midterm test. It is necessary to have a pass grade (PL2 ≥ 5) to pass this test.
- Final exam: It will cover all the course materials. The exam will consist of two parts (P1 and P2). P1 is to be taken by those students that have not passed the first midterm test or for those who, having passed the first midterm test, want to give up their grade. In this last case the grade P1 will substitute the grade PL1. P2 is to be taken by those students that have not passed the second midterm test or for those who, having passed the second midterm test, want to give up their grade. In this last case the grade P2 will substitute the grade PL2.
- The final grade for the theory part (TH) will be the average of the grades obtained in part 1 (P1 or PL1) and part 2 (P2 or PL2). The grades of both parts must be at least 5 to pass the course.
- All the students must take at least two thirds of the evaluable tests (including the lab assignments) to obtain a numerical evaluation. Otherwise they will receive a “No evaluado” grade.
- Whenever a student does not accomplish with any of the requirements for the continuous assessment method, she will be assessed with the non-continuous method.

NON-CONTINUOUS ASSESSMENT METHOD WITHOUT MANDATORY CLASS ATTENDANCE

- Attendance to the lectures is recommended, but not mandatory.
- The grade of the theory part (TH) will be that obtained in the final exam.

The grade for the lab assignments (LAB) can be obtained in two different ways:

CONTINUOUS ASSESSMENT METHOD WITH MANDATORY CLASS ATTENDANCE

- The students must attend to at least 85% of the lab sessions.
- The students must deliver the lab reports respecting the due dates and the lab regulations.
- The lab grade (LAB) will be a weighted average of the grades obtained in each lab. The number of labs and the corresponding weights will be published in the moodle page at the beginning of each course.

NON-CONTINUOUS ASSESSMENT METHOD WITHOUT MANDATORY CLASS ATTENDANCE

- The students must deliver all the lab reports, take a written exam (CF) and do a practical lab test (PP). The final lab grade (LAB) will be obtained as:

CFP = 0,5*CF + 0,5*PP

ATTENTION: Copying is a serious breach of ethics and will punished with severity.

3.2. Resit

EXTRAORDINARY EVALUATION

In the extraordinary exam period:

- The grade for the theory part (TH) will be determined by a final exam.
- The grade for the lab assignments (LAB) will be obtained as for the non-continuous assessment during the ordinary period.

ATTENTION: Copying is a serious breach of ethics and will punished with severity.

4. Proposed workplan

All the course information, including a detailed schedule and syllabus, will be available on the moodle page: http://moodle.uam.es

Week	Content	Lecture hours	Self-study

			hours
1	Introduction. Review of C programming.	1	3
2	**Unit 1: Abstract Data Types**.	3	3
3	**Unit 2: Stacks and applications (2.1-2.3).**	3	2
4	**Unit 2: Stacks and applications (2.4)**.	3	2
5	**Unit 3: Queues and applications (3.1-3.4).**	3	2
6	**Unit 4: Lists (4.1-4.3.1).**	3	2
7	**Unit 4: Lists (4.3.2-4.5).** First midterm test (PL1)	3 2	2
8	**Unit 5: Trees (5.1-5.4).**	3	2
9	**Unit 5: Trees (5.5, 5.6).**	3	2
10	**Unit 6: Priority queues and heaps (6.1, 6.3.1).**	3	2
11	**Unit 6: Priority queues and heaps (6.3.2-6.4).**	3	2
12	**Unit 7: Recursion (7.1-7.4)**	3	2
13	**Unit 7: Recursion (7.5)**	3	2
14	**Recap and exam preparation** Second midterm test	3 2	2 -
	Lab test **Final exam**	4	10
	Extraordinary final exam	3	

The labs schedule will be published in the moodle page at the beginning of the course.