UNIVERSITÀ DEGLI STUDI DI PALERMO

DEPARTMENT	Matematica e Informatica		
ACADEMIC YEAR	2019/2020		
BACHELOR'S DEGREE (BSC)	COMPUTER SCIENCE		
INTEGRATED COURSE	PROGRAMMING AND LABORATORY - INTEGRATED COURSE		
CODE	05880		
MODULES	Yes		
NUMBER OF MODULES	2		
SCIENTIFIC SECTOR(S)	INF/01		
HEAD PROFESSOR(S)	ROCCHESSO DAVIDE	Professore Ordinario	Univ. di PALERMO
OTHER PROFESSOR(S)	LO BOSCO GIOSUE'	Professore Associato	Univ. di PALERMO
	ROCCHESSO DAVIDE	Professore Ordinario	Univ. di PALERMO
CREDITS	12		
PROPAEDEUTICAL SUBJECTS			
MUTUALIZATION			
YEAR	1		
TERM (SEMESTER)	Annual		
ATTENDANCE	Not mandatory		
EVALUATION	Out of 30		
TEACHER OFFICE HOURS	**LO BOSCO GIOSUE'** Tuesday 15:00 17:00 Ufficio al secondo piano del Dipartimento di Matematica e Informatica, Stanza 203. E' suggerita la prenotazione **ROCCHESSO DAVIDE** Thursday 11:00 13:00 Office at first floor of Department of Mathematics and Computer Science. Room 110. Booking is requested. / Ufficio al primo piano del Dipartimento di Matematica e Informatica, Stanza 110. E' suggerita la prenotazione.		

DOCENTE: Prof. DAVIDE ROCCHESSO

PREREQUISITES	No prerequisites.
LEARNING OUTCOMES	Knowledge and understanding: acquisition of the fundamentals of structured programming; elementary static and dynamic data structures; simple fundamental algorithms of searching and sorting; recursive definition of solutions; familiarity with the fundamental constructs of C programming language. Ability to apply knowledge and understanding: ability to tackle simple computational problems by choosing appropriate data structures and algorithms; ability to code in C language; ability to validate, by writing simple programs, the theoretical notions; ability to understand the errors raised by the C compiler or at runtime; ability to decompose complex problems in computationally simpler problems. Critical awareness: ability to choose the most appropriate parameter passing method; ability to compare two simple programs in terms of computational efficiency and behavioral invariance; ability to find an efficient algorithmic solution to simple problems. Communication skills: ability to explain the fundamentals of programming languages and to communicate imperative programs. Learning skills: ability to approach a variety of programming languages and to put the acquired skills at work in concrete problems.
ASSESSMENT METHODS	The exam is composed by a written test, a practical test, and a brief interview. The written test is divided into two parts, each corresponding to one of the two modules. It is made of ten multiple-choice quizzes and of two programming exercises. Each answer is assigned 2 points (correct), 0 points (no answer), or -0.5 points (wrong answer). Programming exercises are assigned up to 6 points each. The written test is considered to be passed if it is scored 15 or more. The practical test is the development of a simple program concerning lists, arrays or file manipulation. The program is evaluated by verifying that it runs and solves the problem, as well as by analyzing the code. It is followed by a brief interview with questions aimed at assessing the knowledge of the main programming notions. The whole practical/oral test is evaluated as a whole, out of 30. The final score is obtained by combining the scores of the written test and of the practical/oral test. The score ranges are qualified as follows: 18-20: knowledge of the subject and programming skills are sufficient; 21-23: knowledge of the subject and programming skills are fair; 24-25: knowledge of the subject and programming skills are good; 26-27: knowledge of the subject and programming skills are very good; 28-30: knowledge of the subject and programming skills are excellent. The "cum laude" can be assigned to those students who have passed the written test with score 26 or higher, and who master the subject and show excellent programming skills, as demonstrated in the practical test and in the interview. An intermediate test is provided at the end of the first module, with a structure similar to the written tests of regular sessions. The student who passes the intermediate test may skip the first part of the written test in the regular session.
TEACHING METHODS	Lectures in the classroom and in the laboratory.

MODULE
ABSTRACT DATA STRUCTURES

Prof. DAVIDE ROCCHESSO

SUGGESTED BIBLIOGRAPHY

P.J. Deitel and H.M. Deitel. Il linguaggio C: Fondamenti e tecniche di programmazione, 7/Ed. Pearson, 2016. (programmazione strutturata in C / structured programming in C)
R. Sedgewick. Algoritmi in C, 4/Ed. Pearson, 2015. (strutture astratte di dati / abstract data structures)

AMBIT	50168-Formazione informatica di base
INDIVIDUAL STUDY (Hrs)	94
COURSE ACTIVITY (Hrs)	56

EDUCATIONAL OBJECTIVES OF THE MODULE

The module aims to provide students with the theoretical and practical tools for the design of iterative and recursive strategies for solving problems. The course focuses on dynamic memory management and on the construction of dynamic data structures, with specific attention to the use of pointers. Recursive and iterative functions are implemented, in different versions and with varying complexity, for the construction and management of data structures, thus allowing the student to familiarise with programming, algorithms, and computational complexity. The construction of data structures at increasing levels of abstraction, is presented and exemplified.
One third of the lecture hours takes place in a classroom equipped with computers, in such a way that the students can directly implement the examples and the exercises proposed by the instructor. This teaching method is suitable for an effective transmission of theoretical and practical skills.

SYLLABUS

Hrs	Frontal teaching
5	Recursion. Examples of recursive functions: factorial of a number, sum of consecutive integers, Fibonacci numbers. Comparison between iteration and recursion. Tail recursion. Introduction to dynamic programming. Recursive functions on arrays: recursive binary search and mergesort algorithm. Computational comparison between mergesort and elementary sorts.
5	Examples of recursive strategies to solve computational problems on arrays and strings. Debugging and profiling C programs. Pointers and dynamic objects. Memory allocation and deallocation. The principle of least privilege.
5	Function pointers. Examples: bidirectional sorting, menus. Structures and derived data types. Operations on structures. Stuctures and pointers. Structures and functions. Dynamic programming: The 0-1 knapsack problem, most profitable path between NW and SE in a matrix. Unions and space sharing.
5	Nodes for dynamic structures. Linked lists. Example: sorted list of characters, insertion and deletion. Comparison between arrays and lists. Examples: recursive inversion of a linked list, search in a linked list. Merging two sorted lists. Mergesort on linked lists. Bottom-up mergesort on arrays. Bottom-up mergesort on linked lists. Natural mergesort.
5	Stack data structure and implementation using a linked list. Queue data structure and implementation using a linked list. Abstract Data Type (ADT): need, features, and definition. The concept of type abstraction and its realization. The stack as an ADT: implementation with linked list, array, or resizable array (dynamic reallocation). Information hiding using the static qualifier.
5	The queue as an ADT: implementation with linked list, array, or resizable array. Circular buffer and buffer resizing. Bottom-up mergesort with a queue of lists. Postfix expression evaluation using a stack. Compilation from multiples source files and use of the pre-processor. Static libraries.
5	Bitwise operations. Bit fields. Enumeration constants and enumerated types. Example: Extraction and representation of the knapsack content. Trees and Binary Search Tree (BST) ADT. BST insertion. Tree traversals. Searching in a BST. Printing a tree. Deletion in a BST.
5	Binary random access files. Queue ADT based on binary random access files. First-class abstract data type (supporting multiple instances). Example: complex number ADT. The queue as a first-class ADT. Systems of queues. The stack as a first-class ADT. Example: infix complex-number expression evaluator. ADT polynomial.

Hrs	Workshops
2	Examples of iterative and recursive functions. Measuring execution time with time.h. Profiling with gprof. Converting the binary search from iterative to recursive form.
2	Recursive mergesort. Switching to insertion sort for small sizes. Measuring performance. Exercises: ordered lists of characters; recursively printing an inverted list; recursive search in a list; fusion of sorted lists.
2	Exercises: deck of cards; recursive knapsack problem; dynamic programming for the recursive knapsack problem; iterative bottom-up knapsack problem

2	List-based implementation of a stack. List-based implementation of a queue. Top-down mergesort with lists. Bottom-up natural mergesort with lists and auxiliary array. Measuring performance.
2	List-based Stack ADT. Array-based Stack ADT. Resizing an array-based stack. Realloc. Verifying element presence in a stack. Abstracting the input-output from the base type.
2	Queue ADT with lists, arrays, and resizable arrays. Circular buffer. List-based mergesort with auxiliary queue. Stability of mergesort. Postfix expression evaluator.
2	Bitwise operations. Printing the bits of an unsigned int. Examples of logic operations and bit shifting. Example: giving back the knapsack content. Bit fields. Enumeration constants.
2	Binary trees. Binary Search Trees (BST). Visits: pre-order, post-order. Exercise: BST search, complexity and performance. Exercise: printing a BST. Exercise: level-order visit and its realization with Queue ADT. BST deletion. ADT polynomial. Infix expressions on complex numbers.

MODULE
STRUCTURED C PROGRAMMING

Prof. GIOSUE' LO BOSCO

SUGGESTED BIBLIOGRAPHY

P. Deitel, H. Deitel. Il linguaggio C. Fondamenti e tecniche di programmazione. Pearson.
Per Consulazione:
K. N. King. Programmazione in C. Apogeo.
A. Bellini, A.Guidi. Linguaggio C - guida alla programmazione. Mc Graw Hill.

AMBIT	50168-Formazione informatica di base
INDIVIDUAL STUDY (Hrs)	94
COURSE ACTIVITY (Hrs)	56

EDUCATIONAL OBJECTIVES OF THE MODULE

The module aims to provide students with the theoretical and practical tools for designing a computer program in its fundamental aspects: the representation of data in structures and the formulation of simple algorithms that adopts the fundamental control, selection and iteration structures. The used programming language is C, due to its diffusion and its characteristic of being preliminary to most of the modern programming languages. One-third of the expected hours of lectures will be held in a classroom equipped with computers so that students can directly carry out examples and exercises proposed by the teacher. This teaching method allows the transfer of theoretical skills into practical examples of implementation.

SYLLABUS

Hrs	Frontal teaching
4	Introduction to the Programming course. The architecture of a computer according to Von Neumann. Representation of information. The binary representation of integers, relative integers, reals, characters. The ASCII code. The strings.
4	Base conversion for the representation of integers. Definition of Algorithm. Examples of algorithms. Flowcharts, representation of an algorithm through a flowchart. Overview of the computational complexity of an algorithm.
4	The C language. Structure of a C program. Compilation, linking, preprocessor. Basic libraries. Identifiers. The constants and the variables. Declaration and assignment. The integer type, float and double. The type char. Basic Input / output functions.
6	Operators in C: arithmetic, relational, logical, bitwise. Order of priority of the operators. Selection constructs in C: If, then, else, and switch, case. Macro. Conditional compilation.
4	The iteration constructs in C. The statement "for". The iteration constructs while, do..while. Equivalence of iteration constructs.
5	Array in C. Static declaration of an array. C strings as static arrays of characters.
5	Pointers. Pointers arithmetic. Dynamic declaration of an array. Stack and Heap. C strings as dynamic character arrays.
6	Multiple-sized arrays. Static and dynamic declaration of a multidimensional array. Equivalence between multidimensional and one-dimensional structures and their representation in memory. The functions in C. The declaration, definition and call of functions. Passing parameters by value and by address.
2	The files in C. Binary and text files. Functions for reading and writing into a file. Random and sequential access.

Hrs	Workshops
2	Coding in C language of first simple programs with selection constructs. Compilation and linking with GCC. Using the make and makefiles command.
2	Implementation in C of programs that use iterative constructs.
2	Implementation in C of programs for inserting and displaying a static array. C implementation of programs to find the length of a string, to compare two strings, to search for a substring.
2	C implementation of programs for inserting and displaying a dynamic array. C implementation of programs to find the length of a string, to compare two strings, to search for a substring using character pointers.
3	Implementation in C of programs for the insertion and visualization of static and dynamic multiple-sized arrays. Implementation in C of the computation of the determinant of a square matrix, by static and dynamic arrays.
3	Implementation in C language of the sorting algorithms, in particular, Insertion and Selection Sort. Implementation in C language of the search for an element in unordered and ordered arrays.
2	Implementation in C language of a program to read and write to a textual or binary file.