Universitatea Naţională de Ştiinţă şi Tehnologie Politehnica Bucureşti
Facultatea de Electronică, Telecomunicaţii şi
Tehnologia Informaţiei

COURSE DESCRIPTION

1. Program identification information

1.1 Higher education institution	**National University of Science and Technology Politehnica Bucharest**
1.2 Faculty	**Electronics, Telecommunications and Information Technology**
1.3 Department	**Electronic Devices, Circuits and Architectures**
1.4 Domain of studies	Electronic Engineering, Telecommunications and Information Technology
1.5 Cycle of studies	Bachelor/Undergraduate
1.6 Programme of studies	Microelectronics, Optoelectronics and Nanotechnologies

2. Date despre disciplină

2.1 Course name (ro) (en)				Structuri de date şi algoritmi Data structures and algorithms (Abstract Data Types)			
2.2 Course Lecturer				Asst./Lect. Dr. Vlad-Alexandru Grosu			
2.3 Instructor for practical activities				Asst./Lect. Dr. Vlad-Alexandru Grosu			
2.4 Year of studies	2	2.5 Semester	1	2.6. Evaluation type	E	2.7 Course regime	Ob
2.8 Course type		D	2.9 Course code	04.D.03.O.005		2.10 Tipul de notare	Nota

3. Total estimated time (hours per semester for academic activities)

3.1 Number of hours per week	3.5	Out of which: 3.2 course	2	3.3 seminary/laboratory	1.5
3.4 Total hours in the curricula	49	Out of which: 3.5 course	28	3.6 seminary/laboratory	21
Distribution of time:					hours
Study according to the manual, course support, bibliography and hand notes Supplemental documentation (library, electronic access resources, in the field, etc) Preparation for practical activities, homework, essays, portfolios, etc.					43
Tutoring					4
Examinations					4
Other activities (if any):					0

3.7 Total hours of individual study	51.00
3.8 Total hours per semester	100
3.9 Number of ECTS credit points	4

4. Prerequisites (if applicable) (where applicable)

4.1 Curriculum	• Computer Programming and Programming Languages 1 (PCLP1) • Algebra and Mathematical Analysis (necessary for understanding the specific properties and modeling of the dynamic data collections studied: stacks, trees, hash tables, graphs)

4.2 Results of learning	Applying the basic knowledge, concepts, and methods from algebra and PCLP1

5. Necessary conditions for the optimal development of teaching activities (where applicable)

5.1 Course	Provision of an optical projector (video projector) together with all related accessories (power, data and video signal cables, remote control).
5.2 Seminary/ Laboratory/Project	• Classroom equipped with personal-computer workstations (desktop/laptop) with the necessary software installed (operating system and the specific working applications required – integrated development environment). • Mandatory attendance at laboratory activities (requirement in accordance with the internal regulations in force regarding activities at POLITEHNICA Bucharest).

6. General objective *(Reffering to the teachers' intentions for students and to what the students will be thought during the course. It offers an idea on the position of course in the scientific domain, as well as the role it has for the study programme. The course topics, the justification of including the course in the currcula of the study programme, etc. will be described in a general manner)*

- Course:

- Understanding the specialized jargon used in the context of dynamic data collections;
- Gaining practical command of the notions required by programming languages, with emphasis on: algorithm, computation routine, dynamic allocation, own library—together with their usefulness;
- Becoming familiar with the modularization techniques provided by the ANSI/ISO C language;
- Developing algorithmic thinking and practicing it through analysis of the studied algorithms and synthesis of new algorithms;
- The ability to comparatively evaluate different algorithms for the same problem and choose the best one, in order to design programs optimized for each real-world situation.

- Laboratory:

- Developing general routines for typical programming-domain applications using *dynamic data collections*;
- Using the ISO C language to design programs;
- Studying aspects related to program portability;
- Interfacing and working directly with most development/design packages that are based on the ISO C/C++ tandem.

7. Competences *(Proven capacity to use knowledge, aptitudes and personal, social and/or methodological abilities in work or study situations and for personal and proffesional growth. They refflect the empolyers requirements.)*

Specific Competences	• C3: Applying basic knowledge, concepts, and methods regarding computer architecture, microprocessors, microcontrollers, programming languages and techniques. • C4: Using programming technologies and environments

Transversal (General) Competences	• CT1: Methodical analysis of problems encountered in activity, identifying elements for which established solutions exist, thus ensuring the fulfillment of professional tasks; • CT3: Adapting to new technologies, professional and personal development through continuous training using printed documentation sources, specialized software, and electronic resources in Romanian and at least one widely used international language.

8. Learning outcomes *(Synthetic descriptions for what a student will be capable of doing or showing at the completion of a course. The learning outcomes reflect the student's acomplishments and to a lesser extent the teachers' intentions. The learning outcomes inform the students of what is expected from them with respect to performance and to obtain the desired grades and ECTS points. They are defined in concise terms, using verbs similar to the examples below and indicate what will be required for evaluation. The learning outcomes will be formulated so that the correlation with the competences defined in section 7 is highlighted.)*

Knowledge	*The result of knowledge aquisition through learning. The knowledge represents the totality of facts, priciples, theories and practices for a given work or study field. They can be theoretical and/or factual.* • **Enumerates** the main classes to which the studied data collections belong. • **Understands** that implementation is not the only core activity in the context of computer modeling of encountered problems. • **Analyzes** design requirements and the steps necessary for designing an arbitrary application. • **Is able to describe** the component areas of the implemented program required to solve a given problem (presented in natural language).
Skills	*The capacity to apply the knowledge and use the know-how for completing tasks and solving problems. The skills are described as being cognitive (requiring the use of logical, intuitive and creative thinking) or practical (implying manual dexterity and the use of methods, materials, tools and intrumentation).* • **Identifies and describes** the basic concepts of dynamic data collections in a given context, together with their implementation methods, for each collection class. • **Models** the proposed problems by formalizing the problem statement (practices the program/application design stage). • **Conceives** new data types and uses them to solve problems related to dynamic data collections. • **Practices** the skills of abstraction and translating into a programming language the ideas expressed (contained) in the problem statement (expressed in natural language). • **Identifies solutions** and **develops** plans for solving the proposed problems (expressed in natural language). • **Analyzes and compares** the solution found for the proposed problem with the suggestions offered by the lab instructor for the specific problem to be solved. • **Tests, debugs, and runs** the written program (the implemented application) and is able to fix possible errors after identifying them beforehand, along with possible consequences (especially semantic).

Responsability and autonomy	*The student's capacity to autonomously and responsably apply their knowledge and skills.* • **Demonstrates receptiveness** to new learning contexts, starting from the *integrative nature* of the subject, based on dynamic data collections. • **Respects** the principles of academic ethics, understanding the responsibilities assumed for individually solving the proposed problems using the learned methods and techniques. • **Demonstrates autonomy** in organizing the learning situation/context or the problem situation to be solved. • **Understands** the implications of the code they write, **learns** how to review others' code, and values the feedback received. • **Is aware** of the value of their contribution in the field of software engineering, from the perspective of professional life, by identifying and proposing their own solutions to solve problems from social and economic life (social responsibility).

9. Teaching techniques *(Student centric techniques will be considered. The means for students to participate in defining their own study path, the identification of eventual fallbacks and the remedial measures that will be adopted in those cases will be described.)*

Interactivity with students through the applied component associated with the taught methods (link to the laboratory).

Time slots are reserved for presenting and solving practical problems (encountered in real life). The modeling part is often reduced to announcing the principles for solving typical programming problems, which require immediate results.

As digital support, the presentations use both the recommended platforms (LMS - Moodle; interactive team management - MSTeams), as well as other variants traditionally considered suitable for supporting teaching activities (e.g., programming support sites, content-creation sites, professional forums).

The human brain has limitations related to multitasking: it offers the capacity to perform 4 things at once. Program modularity and code abstraction are essential objectives, which is why all designed programs are modularized and use the specific C/C++ techniques for isolating the implementation from the programming interface. Finding the appropriate (logically) hierarchy is a fundamental step toward designing a complex system.

Given the common programming core to which the subject belongs (it appears in year 2 of the curriculum), the formation of abstract thinking and conceptualization skills depends largely on the style of the documentation (the audience's experience being limited at this moment). From this perspective, the indispensable documentation provided is easy to use—both in physical format, through the suggested bibliography, and in electronic format (posted using LMS - Moodle).

10. Contents

COURSE		
Chapter	**Content**	**No. hours**
1	= Introduction = Algorithms and their performance. Measuring algorithm running times. Computing the value of a polynomial (classic variant vs. Horner's scheme). Practical usefulness of dynamic data collections.	2

2	= C Program Modularization = Program components. Program module. Collection of programs – the Project concept in writing large-scale programs.	2
3	= Structures and pointers = Functions with structure arguments and with structure return type. Generic (user) data types. Modeling the node in C/C++. Lists: generalities. The concept of linking, particularities. Usefulness.	4
4	= Singly linked lists = Singly linked lists. Typical operations: traversal, insert element, delete element, check empty list, list length. Implementation suggestions.	2
5	Doubly linked lists Usefulness and specific concepts. Typical operations: flexible traversal, insert node, delete node, search element, empty-list test, list length computation. Implementation suggestions.	2
6	= Circular lists = Description. Construction particularities. Representation. Usual operations: insert, delete, traverse, empty-list test. = Stack (special case of SLL) = Operating principle (LIFO). Usefulness, concepts and specific terms. Graphic representation. Specific operations: push, pop, traverse, empty-stack test, top inspection. Implementation suggestions.	4
7	= Queue (special case of SLL) = Operating principle (FIFO). Usefulness, concepts and specific terms. Graphic representation. Specific operations: enqueue, dequeue, inspect element, compute length, traverse, destroy queue, empty-queue test. Implementation suggestions.	2
8	= Hash tables = Working principle and specific concepts. Hash function. Direct addressing. Collision. Load factor. Classification (chaining, open addressing). Usefulness. Types of hash functions (division method, multiplication method, string-dedicated hash functions). Specific operations: insert element, search element, delete element.	3
9	= Trees (general) and binary trees (specialization) = Usefulness. Terms and specific concepts: node, tree degree, branches, path and its length, height of a node, tree height, node level. Graphic representation (accepted conventions): membership, nested parentheses, hierarchical (traditional) form. Algorithmic modeling: downward references, upward references, child–sibling. = Binary trees = Definition and characteristics. Usefulness. Modeling in language. Typical operations: insert node, delete node, traversal (depth, breadth), search information, compute height. Creating binary trees. Balanced binary tree: construction method. Traversing binary trees (and balanced ones): in-order, pre-order, post-order. Level-order (breadth) traversal. Compute height for binary trees. Searching for information in a tree.	4

10	= Introduction to graphs = Basic concepts, representation techniques, traversal and implementation variants. Applicability to practical problems.	2
11	Review: examples of exercises and problems.	1
	Total:	28

Bibliography:

1. Şl. Dr. ing. Vlad-Alexandru GROSU – Structuri de Date şi Algoritmi, suport de curs în format electronic (Moodle) (https://curs.upb.ro/2024/course/view.php?id=3374, https://curs.upb.ro/2023/course/view.php?id=10043)
2. Leon LIVOVSCHI, Horia GEORGESCU – *Sinteza şi analiza algoritmilor*, Ed. Ştiinţifică şi Enciclopedică, Bucureşti, 1986.
3. Florin MUNTEANU, Gh. MUSCĂ, Florin MORARU – *C - Tehnici de programare*, Ed. Joint Printing House, 1995.
4. Emanuela CERCHEZ, Marinel ŞERBAN – *Programare în limbajele C/C++ pentru liceu*, vol. 3, Polirom, 2006.
5. Thomas CORMEN, Charles LEISERSON, Ronald RIVEST – *Introducere în algoritmi*, Computer Libris, Agora, Cluj (2000-2009).
6. Kyle LOUDON – *Mastering algorithms with C*, O'Reilly, 1999.
7. Robert SEDGEWICK, Kevin WAYNE – *Algorithms*, 4th ed., Addison Wesley, 2011.
8. Niklaus WIRTH – *Algorithms + Data structures = Programs*, Prentice Hall, ≥ 1995.
9. Ellis HOROWITZ, Sartaj SAHNI – *Fundamentals of computer algorithms*, Springer, 1983-1998.

LABORATORY

Crt. no.	Content	No. hours
1	Singly linked lists. Designing a dedicated library.	2
2	Doubly linked lists. Designing a dedicated library.	2
3	Intermediate testing and evaluation of solutions.	2
4	Hash tables. Complete software solution.	2
5	Binary search trees. Complete software solution.	2
6	Intermediate testing and evaluation of solutions.	2
7	Examples of solved problems (lists, stacks, queues)	2
8	Examples of solved problems (hash tables)	2
9	Examples of solved problems (binary trees)	2
10	Final laboratory check (colloquium). Identifying the real context presented in the problem and practical adaptation (through implementation) of the studied concepts.	2
11	Programming skills test.	1
	Total:	21

Bibliography:

1. Şl. dr. ing. Vlad-Alexandru GROSU – Structuri de Date şi Algoritmi, materiale studiu laborator site personal (https://itlectures.ro/ro_RO/sda/materiale-studiu-sda/) + Moodle: (https://archive.curs.upb.ro/2021/user/files.php#)
2. I. Rusu, Dana Gavrilescu, Vlad Al. Grosu – *Îndrumar de laborator pentru programarea calculatoarelor: C*, Editura MatrixRom, Bucureşti, 2004.
3. Florin Munteanu, Gh. Muscă, Florin Moraru – *C - tehnici de programare*, Editura Joint Printing House, Bucureşti, 1995.
4. Niklaus WIRTH -– *Algorithms + Data structures = Programs*, Prentice Hall, 1995.
5. https://linux.die.net (site suport pentru API-ul C/C++)
6. https://stackoverflow.com/ (forum dedicat discuţiilor practice din zona IT şi conex)

11. Evaluation

Activity type	11.1 Evaluation criteria	11.2 Evaluation methods	11.3 Percentage of final grade
11.4 Course	- Correctly identifying the theoretical and practical contexts for applying the taught algorithms. - Deepening the language notions required by the course, with application in the field of computer engineering and information technologies (as fundamental training areas). C4.1: Defining the concepts, principles, and methods used in the domains of: computer programming, high-level and specific languages, computer system architecture, programmable electronic systems, graphics, reconfigurable software architectures.	- Scheduled verification paper in the pre-session. - Both the students' ability to formalize (identify the notions and concepts specific to the studied language implied by the respective problem) and to translate into language (program implementation) problems expressed in Romanian (natural language) are investigated. - The topics cover both the theoretical part regarding the definition of concepts of dynamic data collections, and the practical part, from the perspective of the ability to solve, using ANSI C/C++ (through algorithmization), typical programming problems (presented in Romanian).	40%

11.5 Seminary/laboratory/project	C4.5: Supporting and passing a test regarding the architecture and practical principles of a functional software structure. C6.5: Supporting a test on establishing and describing the operations necessary to implement and test an algorithm.	Laboratory activity is constantly checked throughout the semester. Students can accumulate 30%, following two in-term verification tests, each of 30 minutes. The laboratory ends with an individual final verification (colloquium) at the workstation (weight: 30%). This is based on the implementation of an algorithm from the material covered during the entire semester and on solving a theoretical topic with a synthetic role.	60%
11.6 Passing conditions			
• Verification of the abilities of *understanding and identifying* the real practical situations specific to the presented methods, as well as the correct application thereof in the IT/ engineering and information technologies area, through a practical (individual) implementation test, by which the capability is tracked of abstraction and conceptualization in the terms of a programming language and then choosing the most suitable dynamic collections according to the specifications, expressed in natural language, of a real problem. • Passing the subject requires accumulating (with no imposed intermediate thresholds) ***at least 50p*** out of the total of 100p.			

12. Corroborate the content of the course with the expectations of representatives of employers and representative professional associations in the field of the program, as well as with the current state of knowledge in the scientific field approached and practices in higher education institutions in the European Higher Education Area (EHEA)

At present, it is necessary to train future engineers and researchers in algorithms for solving practical problems of software design and testing, database administration, or even those forming the basis of an operating system's kernel. Training students on the programming core provides the fundamental knowledge needed for any subsequent professional activity: teaching, research and/or design.

The activities of the subject *Data Structures and Algorithms* provide the foundations of algorithmic thinking and of the techniques specific to applying a standardized programming language (programming is done in ISO C/C++) in the context of practical real-life problems and also in the context of computing systems (algorithms encountered within operating systems). Starting from the title of a reference book in the field, "*Algorithms + Data Structures = Programs*" written by Niklaus Wirth, a serious program cannot be conceived without implementing *dynamic data collections* ("abstract data types"). The course's purpose is to set at least the basics for understanding and using these data collections, training in program writing and absolutely necessary syntactic principles already known from PCLP1/2, such as memory management in a program—through *dynamic allocation* techniques.

Students acquire the use of an integrated programming environment. The variants used in the laboratory (DevC++, Code::Blocks) are based on the MinGW (x86 or x64) port of the open-source tools (gcc, g++, gdb, gprof, make) from the Unix/Linux world to the closed-source Windows world. In the offered variants, they support the minimal C language standard (ISO/IEC 9899:1999, known as C99), respectively the minimal C++ standard (ISO/IEC 14882:2011, called C++11).

By structuring the information according to the activities provided in the curriculum, as well as through the tutoring activity carried out, the subject offers the necessary steps for assessing the quality, merits, and limits of processes, programs, projects, concepts, methods, and theories. With an engaged study of the subject, students will be able to discern (choose) among the various algorithms needed in professional practice.

The proper use of evaluation criteria and methods, in accordance with the European academic norms to which POLITEHNICA University of Bucharest is a party, allows students to continuously self-evaluate (‘personal feedback’), starting from the grades obtained and taking into account the observations and methodological indications offered by the course/lab holder.

Date	Course lecturer	Instructor(s) for practical activities
25.09.2025	Asst./Lect. Dr. Vlad-Alexandru Grosu	Asst./Lect. Dr. Vlad-Alexandru Grosu

Date of department approval	Head of department
26.09.2025	Prof. Dr. Claudius Dan

Date of approval in the Faculty Council	Dean
26.09.2025	Prof. Dr. Mihnea Udrea