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's
1.6 Programme of studies	Microelectronics, Optoelectronics and Nanotechnologies

2. Date despre disciplină

2.1 Course name (ro) (en)				Programarea calculatoarelor şi limbaje de programare 1 Computer Programming and Programming Languages 1			
2.2 Course Lecturer				Lect. Dr. Vlad-Alexandru Grosu			
2.3 Instructor for practical activities				Lect. Dr. Vlad-Alexandru Grosu / Teaching Assistant (PhD student) Grigore Ţiplea			
2.4 Year of studies	1	2.5 Semester	I	2.6. Evaluation type	E	2.7 Course regime	Ob
2.8 Course type		F	2.9 Course code	04.F.01.O.004		2.10 Tipul de notare	Nota

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

3.1 Number of hours per week	4	Out of which: 3.2 course	2	3.3 seminary/laboratory	2
3.4 Total hours in the curricula	56	Out of which: 3.5 course	28	3.6 seminary/laboratory	28
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.					38
Tutoring					2
Examinations					4
Other activities (if any):					0

3.7 Total hours of individual study	44.00
3.8 Total hours per semester	100
3.9 Number of ECTS credit points	4

4. Prerequisites (if applicable) (where applicable)

4.1 Curriculum	• Algebra and Mathematical Analysis • Mathematical Logic
4.2 Results of learning	Applying basic knowledge, concepts, and methods from algebra and mathematical analysis to computer-based modeling of the problems proposed for solution.

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

5.1 Course	It will take place in a room equipped with a video projector and a computer, together with all related accessories: power cables (data and video signal), and the projector remote control.
5.2 Seminary/ Laboratory/Project	• It will take place in a room with specific equipment, which must include: • General-purpose computing systems (personal computers) required to implement specific methods and algorithms, with the necessary software installed (operating system and required working applications—integrated development environment, IDE) • Whiteboard along with specific accessories: special markers, whiteboard eraser, and cleaning solutions.

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)*

The discipline is scheduled in year 1, semester 1 (the common core of engineering training).

It is a fundamental engineering subject that provides the study foundations especially for later specializations based on intensive work with the personal computer, which include in the curriculum fundamental subjects based on the use/design of algorithms, source code writing (implementation) as well as debugging activities (identifying semantic errors).

For the lecture, the aim is to assimilate the concepts of structured programming (with a view to syntactically grounding object-oriented programming in C++, which will follow), necessary for computer modeling of various real-life problems, as well as those involved in designing and implementing dedicated applications or tasks encountered in practice. Various practical situations are analyzed and algorithms are implemented in a high-level language designed using object-oriented software design principles (ISO C). The programs developed in the laboratory help form mental reflexes in modeling reality and assist students in their future engineering activity. The skills and abilities provided by the course constitute fundamental knowledge absolutely necessary for subjects specific to software-oriented specialization tracks that students may choose in the next study cycle (starting with the 3rd year).

For the laboratory (applications), the proposal is to develop general routines of at most medium level dedicated to solving situations encountered in real life. Thus, easy construction by hardware/software designers of a specific library is envisaged. Programs are conceived using the ISO C/C++ languages, thereby ensuring program portability and allowing immediate interfacing and work with most development/design packages that are based on these related languages.

The skills acquired from writing programs in the laboratory (and using the integrated development environment) become working tools for students, first for activities within the specialization years (depending on their choices) and later, for year projects and the bachelor’s thesis, and ultimately in the private sector (future software engineer).

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 system architecture, microprocessors, microcontrollers, programming languages, and programming techniques.
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 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** programming concepts in general and structured programming in particular (per the Böhm–Jacopini theorem) that underlie the studied language. • **Understands** that implementation is not the only core activity in the context of computer programming and, more broadly, software development. • **Analyzes** design requirements and the steps needed to actually design an application (as a consequence of the previous statement). • **Assimilates** the specialized vocabulary (jargon) used in the programming field. • Is able to **describe** the component areas of the implemented ISO C program needed 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 fundamental concepts together with their implementation methods: data/function hiding, code reuse, function overloading and overriding. • **Models** the proposed problems by formalizing the problem text (practices the program/application design stage). • **Identifies** the program components that interact in the application to be written. • **Identifies solutions** and **develops plans** for solving the proposed problems (expressed in natural language). • **Analyzes** and compares the solution found for the assigned problem with the suggestions offered by the lab instructor for that specific problem. • **Tests, debugs, and runs** the written program (the implemented application) and is able to correct possible errors after identifying them in advance, along with possible consequences.
Responsability and autonomy	*The student's capacity to autonomously and responsably apply their knowledge and skills.* • **Demonstrates receptiveness** to new learning contexts (structured programming principles may be new from the curriculum perspective). • **Respects** academic ethics, understanding the responsibilities assumed in individually solving proposed problems using the methods and techniques learned. • **Demonstrates autonomy** in organizing the learning situation/context or the situation posed by the problem to be solved. • **Becomes aware** of the value of their contribution in software engineering, in active life, by identifying and proposing their own solutions to solve problems in 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.)*

Starting from the analysis of students' learning characteristics and their specific needs, identified by the course holder through personal experience in the private sector, the teaching process uses both expository methods (lecture, presentation) and conversational–interactive methods, based on action-based learning models such as exercises or solving programming problems. Interactivity with students through the practical component associated with the taught concepts is particularly emphasized.

Time slots are reserved for presenting and solving current problems that students encounter in year 2 electronics subjects: modeling electronic devices, analyzing passive components and circuits, and preparing engineering skills for problem solving using computers (numerical methods).

The modeling part often reduces to solving practical problems identified in real life. The presented concepts prepare the way for correlated subjects in years 1 and 2 (Computer Programming and Programming Languages 2, Data Structures and Algorithms, Numerical Methods) and later, in years 3 and 4, for subjects such as: Deep Learning and Artificial Intelligence, Robotics, or Neural Networks and Fuzzy Systems.

Dialogue during the lecture continues in the laboratory sessions. These are necessary for preparing students for ongoing individual verification tests, which build skills for solving problems under time constraints. A typical session begins with a brief review of the programming concepts specific to the lab. Afterwards, students aim to design and write complete, functional programs—starting from a statement in natural language (Romanian).

The language used is ISO C, standard C 1999. The open-source development environment used in the lab is configurable accordingly and can be instructed to use the indicated ISO C standard variant via the compilation option -std=c99. The integrated development environment and platform materials for the lab are available to students in electronic form.

10. Contents

COURSE		
Chapter	**Content**	**No. hours**
1	Introduction. The notion of a compiler. C standards. Bibliographic commentary.	2
2	Structure of C programs. Syntax–semantics distinction. Components of a program. Language alphabet. Keywords.	2
3	Memory: regions and address spaces. Memory alignment. Number bases. Types of memory. Memory models used by the compiler. Fundamental data types in C. Declaring and defining variables and constants. Storage classes. Global variables.	2
4	Operators and expressions. Classification of operators. Examples for the identified operator classes. Precedence and associativity. Implicit and explicit conversions. Details about the comment area in the general program structure. Details about header files. Details about the preprocessor directives area. Inline functions (macros): introduction.	2
5	Statements. Expression, block, decision (if), selection (switch), and looping statements. Examples. Jump statements: continue, break, goto.	2
6	Function prototypes and definitions. Inline functions (continued): differences between macro expansion and a function call. The main() function: details. Modularization. Details on the user-defined type area (typedef). Details on the function prototypes area.	2
7	Functions. Declaration and definition. Local and external variables. Function calls. Formal and actual parameters. Stack. Calling conventions. Macro functions (details). Comparing iterative and recursive functions.	2
8	Pointers. Declaration, initialization. Address-of and dereference operators. The special NULL value. Pointer arithmetic. The void type applied to pointers. Functions with pointer arguments and pointer return types. Pointers to functions. Dynamic memory allocation techniques.	4

9	Arrays. 1-D and n-D arrays: declaration, definition. Initialization. Indexing. Strings. The array–pointer relationship. Functions with array arguments. Enumerations: declaration/definition. Enumerations and pointers. Functions with enumeration arguments.	3
10	Unions: declaration/definition, initialization. Non-homogeneous data types. Structures: declaration, definition, initialization. Allowed operations on structures. Structures and pointers. The sizeof operator applied to structures.	3
11	C language I/O. The notions of stream and file. Files. File operations: open, close, write, read, query/set the file position indicator. Physical deletion of files. Renaming. Error-handling functions.	2
12	Advanced aspects of the language. The concept of variable. Declarations/definitions (implementation details). Scope and lifetime of variables. Variable binding. Storage classes and namespaces. R-value/L-value. Compilation stages. Functions with a variable number of arguments. Recursive functions (detail). Creating custom data types: library types.	2
	Total:	28

Bibliography:

1. Lect. Dr. Eng. GROSU Vlad-Alexandru, PCLP1, electronic course support (Moodle): (https://archive.curs.upb.ro/2021/course/view.php?id=8868, https://curs.upb.ro/2024/course/view.php?id=3871)
2. I. Rusu, Dana Gavrilescu, Vlad Al. Grosu - *Programarea calculatoarelor în limbaj C*, Editura MatrixRom, Bucureşti, 2002.
3. I. Rusu, Vlad Al. Grosu – *Programarea calculatoarelor în limbaj C: probleme rezolvate şi comentate*, Editura MatrixRom, Bucureşti, 2008.
4. D.I. Năstac, *Programarea calculatoarelor în limbajul C – Elemente fundamentale*, Editura Printech, Bucureşti, 2006.
5. D. Burileanu, C. Dan, M. Pădure, *Programare în C. Culegere de probleme*, Editura Printech, Bucureşti, 2004.
6. Brian Kernighan, Dennis Richie – *The C Programming Language*, Prentice Hall, New Jersey, 1978 & 1988 editions.
7. Emanuela Cerchez, Marinel Şerban – *Programarea în limbajul C/C++ pentru liceu*, vol. 1, Polirom, ≥ 2013.

LABORATORY

Crt. no.	Content	No. hours
1	Number bases. Conversions between common number bases.	2
2	Language alphabet. Keywords. Declaring and defining variables.	2
3	Statement block. Operators (introduction). Building expressions. Statements.	2
4	Operators (continued): assignment operator. Explicit cast. Sequencing operator. User-defined types: typedef. The switch–case statement. Interrupts and jumps in C. The for loop.	2

5	while and do–while loops. Arrays (introduction).	2
6	Arrays (continued): strings. Static allocation. Pointers (I) – Introduction.	2
7	Pointers (II) – NULL pointer. Pointer arithmetic. Array–pointer relationship.	4
8	Pointers (III): pointers and dynamic allocation.	4
9	Structures. The typedef operator applied to structures. Pointers and structures: building an abstract data type.	2
10	Enumerations. Unions. Differences between unions and structures.	2
11	I/O operations. Library functions used for working with files.	2
12	Final laboratory verification.	2
	Total:	28

Bibliography:

1. Lect. Dr. Eng. GROSU Vlad-Alexandru, PCLP1, electronic course support (Moodle) (https://archive.curs.upb.ro/2021/course/view.php?id=8868)
2. I. Rusu, Dana Gavrilescu, Vlad Al. Grosu - *Programarea calculatoarelor în limbaj C*, Editura MatrixRom, Bucureşti, 2002.
3. I. Rusu, Dana Gavrilescu, Vlad Al. Grosu – *Laboratory Guide for Computer Programming: C*, Editura MatrixRom, Bucureşti, 2004.
4. I. Rusu, Vlad Al. Grosu – *Programarea calculatoarelor în limbaj C: probleme rezolvate şi comentate*, Editura MatrixRom, Bucureşti, 2008.
5. Brian Kernighan, Dennis Richie – *The C Programming Language*, Prentice Hall, New Jersey, 1978 & 1988 editions.
6. Herbert Schildt – *C: The Complete Reference*, Teora, 1999–2003.
7. Florin Munteanu, Gh. Muscă, Florin Moraru – *C – Programming Techniques*, Joint Printing House, Bucharest, 1995.

11. Evaluation

Activity type	11.1 Evaluation criteria	11.2 Evaluation methods	11.3 Percentage of final grade

11.4 Course	- Correctly identifying theoretical and practical contexts for applying the taught algorithms and programming techniques. - Defining the concepts, principles, and methods used in the fields of: computer programming, high-level and domain-specific languages, computer system architecture, programmable electronic systems, graphics, reconfigurable software architectures.	- Sitting the individual written examination (total: 40 points). - Both the students' ability to formalize a problem and to translate specific concepts into a program are assessed. - Topics cover both the theoretical part defining object-oriented programming concepts and the practical part, assessing the ability to solve programming problems (presented in Romanian) using the ISO C language.	40%
11.5 Seminary/laboratory/project	- C4.5: Sitting and passing an assessment on the architecture and functional principles of a working software structure. - C6.5: Sitting an assessment on establishing and describing the operations necessary to implement and test a numerical algorithm.	Laboratory activity is continuously verified throughout the semester via individual computer-based assessments. Students can accumulate: • 10% from activity tests at each lab; • 20% from the midterm test; • 30% from the final colloquium—the laboratory ends with an individual final verification at the workstation. This is based on implementing an algorithm from the semester's material and answering a synthetic theoretical question.	60%
11.6 Passing conditions			
• Skills are checked for accumulating and then identifying practical situations specific to the presented methods, as well as correctly applying these methods in information engineering. • Passing the subject requires accumulating (with no imposed intermediate thresholds) ***at least 50 points*** out of the total 100 points available.			

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)

Currently, training future engineers and researchers in numerical algorithms for solving design, testing, or signal processing problems by various mathematical methods is essential. Training students in programming provides the fundamental knowledge required in any later professional activity: teaching, research, and/or design.

The activities related to Computer Programming provide the foundations of algorithmic thinking and programming in a modern programming language, namely ANSI/ISO C according to its latest standardization from `2011` (`ISO/IEC 9899:2011`).

By structuring information according to the curriculum activities, as well as through the conducted tutoring activity, the subject provides the steps needed to assess the quality, merits, and limits of processes, programs, projects, concepts, methods, and theories.

Appropriate use of evaluation criteria and methods, in line with European academic norms to which the POLITEHNICA University of Bucharest adheres, allows students to self-assess continuously, based on obtained grades and considering the observations and methodological guidance provided by the course/lab holder.

Date	Course lecturer	Instructor(s) for practical activities
18.09.2025	Lect. Dr. Vlad-Alexandru Grosu 	Lect. Dr. Vlad-Alexandru Grosu  Teaching Assistant (PhD student) Grigore Țiplea 

Date of department approval	Head of department Prof. Dr. Claudius Dan
Date of approval in the Faculty Council	Dean Prof. Eng. Dr. Radu Mihnea Udrea

Universitatea Națională de Știință și Tehnologie Politehnica București

Facultatea de Electronică, Telecomunicații și

Tehnologia Informației