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)				Programarea calculatoarelor şi limbaje de programare 2 Computer Programming and Programming Languages 2			
2.2 Course Lecturer				S.l./Lect. Dr. Vlad-Alexandru Grosu			
2.3 Instructor for practical activities				S.l./Lect. Dr. Vlad-Alexandru Grosu			
2.4 Year of studies	1	2.5 Semester	2	2.6. Evaluation type	E	2.7 Course regime	Ob
2.8 Course type		F	2.9 Course code	04.F.02.O.011		2.10 Tipul de notare	Nota

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

3.1 Number of hours per week	3	Out of which: 3.2 course	2	3.3 seminary/laboratory	1
3.4 Total hours in the curricula	42	Out of which: 3.5 course	28	3.6 seminary/laboratory	14
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.					52
Tutoring					2
Examinations					4
Other activities (if any):					0

3.7 Total hours of individual study	58.00
3.8 Total hours per semester	100
3.9 Number of ECTS credit points	4

4. Prerequisites (if applicable) (where applicable)

4.1 Curriculum	Completion and/or passing of the following courses: Computer Programming and Programming Languages 1 (PCLP1)

4.2 Results of learning	Acquisition of the following knowledge: • Applying basic knowledge, concepts, and methods related to computer system architecture, programming languages, and programming techniques; • Solving concrete practical problems that include programming elements and the use of computing systems with microprocessors or microcontrollers.

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

5.1 Course	Held in a room equipped with a video projector and computer, together with all related accessories: power/data/video cables, and projector remote control.
5.2 Seminary/ Laboratory/Project	Held in a room with specific equipment, which must include: • General-purpose computer systems (PCs) required to implement the specific methods and algorithms, with the necessary software installed (operating system and required applicationsâ€”an IDE). • Whiteboard with specific equipment: special markers, eraser, and cleaning solution. Attendance is mandatory for laboratory activities (per UNSTPB internal regulations).

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 course is scheduled in Year 1, Semester 2.

It is a fundamental engineering course that provides the study foundations—especially for later specializations involving intensive PC work—which include courses strongly based on algorithm design and source-code implementation.

For the lecture, the goal is to acquire object-oriented programming concepts needed to model real-life problems and to design and implement dedicated applications and practical assignments. Various practical situations are analyzed and algorithms are implemented in a high-level language designed with object-oriented principles (ISO C++). The programs developed in the lab build mental reflexes for modeling reality and help students in their future engineering work. The skills gained are fundamental for software-oriented specializations that students choose in later study cycles (starting with Year III).

For the lab (applications), we propose developing general-purpose routines for numerical applications—enabling the easy construction of a specific library by hardware/software designers. The use of ISO C/C++ ensures program portability and enables immediate interfacing with most development/design packages that rely on these languages. Topics are based on OOP concepts and principles such as: classes and objects, member functions, constructors, destructors, data hiding, abstraction, friend functions, references and pointers (the *this* pointer), single and multiple inheritance, polymorphism, and overloading.

The skills acquired in writing programs in the lab become working tools for students in year projects and the bachelor’s thesis, as well as in future software engineering activity.

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 architectures, 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 learning using printed 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** the fundamental OO concepts underpinning the studied language. • **Understands** that implementation is not the only core activity in OOP and software development; requirements analysis and application design are also involved. • **Defines** UML (Unified Modeling Language) notions specific to the domain. UML formally presents possible interactions among classes in a program, aligning with the application design stage to be implemented next. • Is able to **describe** the component areas of an implemented 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** basic OO concepts along with their implementation methods: data/function hiding, code reuse, function overloading and overriding. • **Models** proposed problems by formalizing the text (practices the program/application design stage). • **Identifies** the classes (later, objects) that interact in the application to be written. Corollary: **practices** abstraction skills and translation of ideas expressed in natural language into the programming language. • **Identifies** solutions and **develops** plans for solving the proposed problems (expressed in natural language). • **Analyzes and compares** the found solution with suggestions offered by the lab holder for the specific problem. • **Tests, debugs, and runs** the written program (implemented application) and is able to fix possible errors after identifying them and their consequences.
Responsability and autonomy	*The student's capacity to autonomously and responsably apply their knowledge and skills.* • **Demonstrates** receptiveness to new learning contexts (OOP principles constitute a new paradigm, continuing the structured paradigm). • **Respects** academic ethics, understanding the responsibilities of individually solving proposed problems using learned methods and techniques. • **Demonstrates** autonomy in organizing the learning context or the problem-solving situation. • **Becomes aware** of the value of their contribution to software engineering from the perspective of active life, by identifying and proposing their own solutions to social and economic problems (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.)*

Based on students' learning characteristics and specific needs—identified by the course holder through private-sector experience—teaching uses both expository methods (lecture, presentation) and interactive/conversational methods, with action-based learning models such as exercises or solving programming problems. Interactivity via the applied part associated with the taught concepts is emphasized.

Modeling often reduces to solving practical, real-life problems through OO modeling. We present how OO principles are reflected in concrete implementation approaches (the link between OO principles and OO programming practice). The concepts reappear in specialization years (3 and 4) in courses such as Deep Learning and Artificial Intelligence, Robotics, or Neural Networks and Fuzzy Systems.

Dialogue during lectures continues in lab sessions. These are needed to prepare students for ongoing individual checks. A typical session starts with a brief review of the programming notions specific to the assignment (where applicable, with a comparative presentation versus structured programming). Afterwards, students aim to design and write complete, functional programs.

The language used is ISO C++, standard C++11. The open-source development environment used in the lab can be configured accordingly and instructed to use C++11 via the compilation option `-std=c++11`. The IDE and lab platform materials are available to students electronically.

Presentations (lecture + lab) use images and UML diagrams (for the algorithms used—where applicable—and for OO modeling) so that the presented information is easy to understand and assimilate.

10. Contents

COURSE		
Chapter	**Content**	**No. hours**
1	Fundamental concepts of object-oriented programming.	3
2	Characterization of C++ (compared to ANSI/ISO C). Concrete examples using C++ syntax.	2
3	Classes and objects. Static–dynamic distinction in terms of program effects. OO concepts regarding classes: member functions, constructors, destructors, data hiding, abstraction.	2
4	Member functions (methods): functional details. Examples. Data access specifiers.	2
5	Constructors: details. Default, parameterized, copy constructors. Examples. Destructors. Examples. Friend functions. Examples.	3
6	C++ references. The this pointer. Pointers to classes. Static vs. dynamic in this context. Examples.	2
7	Static class members. Static functions.	2
8	Inheritance: introduction. Specific concepts and rules.	3
9	Inheritance: continuation. Single inheritance. Data/method access specifiers. Public, protected, and private inheritance.	2
10	Multiple inheritance. Polymorphism. Function overloading. Early and late binding. Virtual methods.	3
11	Interface inheritance vs. implementation inheritance. Operator overloading. Overloadable operators. Exception handling.	2
12	Review: sample exercises and problems.	2
	Total:	28

Bibliography:

1. Lect. Dr. ing. Vlad-Alexandru GROSU, PCLP2, *electronic course support* (Moodle) (https://curs.upb.ro/2021/course/view.php?id=8979) or OOP: https://curs.upb.ro/2024/course/view.php?id=7978
2. Brian Overland, *C++ – Guide for Beginners*, Corint, Bucharest, 2006.
3. Danny KALEV, Michael TOBLER, Jan WALTER – *C++*, Waite Group, January 1999 (also published by Teora, 2000)
4. Ionuţ MUŞLEA – *Initiation in C++ (Object-Oriented Programming)*, Microinformatica, Cluj, 1993.
5. Dan SOMNEA, Doru TURTUREA – *Initiation in C++ (Object-Oriented Programming)*, Technical Publishing, 1993.
6. Emanuela Cerchez, Marinel Şerban – *Programming in C/C++ for High School*, vol. 4, Polirom, ≥ 2013.

LABORATORY		
Crt. no.	**Content**	**No. hours**
1	Program structure recap. Programming skills check (short quiz). Comparison between procedural (structured) and object-oriented programming.	2
2	Introduction to classes and objects. Practical distinction between the two concepts.	2
3	Classes and objects (cont.). Constructors: parameterized, copy. Friend functions.	2
4	Constructors (cont.). Destructors. Functions with object arguments and references to objects.	2
5	Single inheritance.	2
6	Multiple inheritance. Polymorphism.	2
7	Final lab check	2
	Total:	14

Bibliography:

1. Lect. Dr. ing. Vlad-Alexandru GROSU, PCLP2, electronic course support (Moodle) (https://archive.curs.upb.ro/2021/course/view.php?id=8979)
2. Brian OVERLAND – *C++ – Guide for Beginners*, Corint, 2006.
3. Herbert SCHILDT – *C++ – Complete Reference*, Teora, 2001.
4. Adonis BUTUFEI – Introduction to C++, InfoAcademy Net (https://pdfcoffee.com/curs-infoacademy-c-pdf-free.html)

11. Evaluation

Activity type	11.1 Evaluation criteria	11.2 Evaluation methods	11.3 Percentage of final grade
11.4 Course	Correct identification of theoretical and practical contexts for applying the taught methods and techniques. Definition of concepts, principles, and methods used in: computer programming, high-level and specific languages, computer architecture, programmable electronic systems, graphics, reconfigurable software architectures.	Written individual exam (total: 40 pts). Assesses both the ability to formalize a problem and to translate OO concepts into a program. Subjects cover both the theory of OOP concepts and the practical ability to solve programming problems in C++ (stated in Romanian).	40%

11.5 Seminary/laboratory/project	• (C4.5) Passing an assessment on the architecture and functional principles of a working software structure. • (C6.5) Passing an assessment on defining and describing the operations needed to build and test a specific algorithm, designed using the OOP paradigm.	Laboratory activity is continuously assessed throughout the semester via: • activity quizzes each lab: 10% • midterm verification test (on computer): 20% • colloquium: 30% — the lab ends with an individual final check at the workstation, based on implementing a problem stated in natural language covering the whole semester.	60%
11.6 Passing conditions			
• Checks the abilities to accumulate and then identify practical situations specific to the presented methods, as well as correctly applying these methods in information engineering. • Passing requires accumulating ***at least 50 pts*** out of 100 (no intermediate thresholds imposed).			

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)

It is currently necessary to train future engineers and researchers in numerical algorithms that solve design, testing, or various signal-processing problems using different mathematical methods. Training students on a programming core provides the fundamental knowledge needed in any subsequent professional activity: education, research, and/or design.

The activities related to Computer Programming provide the foundations of algorithmic thinking and programming in a current programming language, namely ISO C++ according to its 2011 standardization (ISO/IEC 14882:2011).

By structuring information according to curriculum activities and through tutoring, the course offers 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—aligned with European academic standards to which the POLITEHNICA University of Bucharest adheres—allows students to continually self-evaluate, based on grades received and considering methodological observations and guidance offered by the course/lab holder.

Date	Course lecturer	Instructor(s) for practical activities

Universitatea Naţională de Ştiinţă şi Tehnologie Politehnica Bucureşti

Facultatea de Electronică, Telecomunicaţii şi

Tehnologia Informaţiei

25.09.2025	S.l./Lect. Dr. Vlad-Alexandru Grosu 	S.l./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 