SYLLABUS

Object Oriented Programming

University year 2025-2026

1. Information regarding the programme

1.1. Higher education institution	Babeş-Bolyai University of Cluj-Napoca
1.2. Faculty	Faculty of Mathematics and Computer Science
1.3. Department	Departament of Computer Science
1.4. Field of study	Computer Science
1.5. Study cycle	Bachelor
1.6. Study programme/Qualification	Computer Science
1.7. Form of education	Full time

2. Information regarding the discipline

2.1. Name of the discipline	**Object Oriented Programming**					Discipline code	**MLE5006**
2.2. Course coordinator					Assoc. Prof. PhD Bocicor Maria Iuliana		
2.3. Seminar coordinator					Assoc. Prof. PhD Bocicor Maria Iuliana		
2.4. Year of study	**1**	2.5. Semester	**2**	2.6. Type of evaluation	**E**	2.7. Discipline regime	**Compulsory**

3. Total estimated time (hours/semester of didactic activities)

3.1. Hours per week	**5**	of which: 3.2 course	**2**	3.3 seminar/laboratory/project	**1 sem 2 lab**
3.4. Total hours in the curriculum	70	of which: 3.5 course	28	3.6 seminar/laboratory/project	**14+28**
Time allotment for individual study (ID) and self-study activities (SA)					**hours**
Learning using manual, course support, bibliography, course notes (SA)					20
Additional documentation (in libraries, on electronic platforms, field documentation)					5
Preparation for seminars/labs, homework, papers, portfolios and essays					19
Tutorship					4
Evaluations					7
Other activities:					
3.7. Total individual study hours	**55**				
3.8. Total hours per semester	**125**				
3.9. Number of ECTS credits	**5**				

4. Prerequisites (if necessary)

4.1. curriculum	Fundamentals of Programming
4.2. competencies	Average programming skills in a high level programming language

5. Conditions (if necessary)

5.1. for the course	Class room with projector
5.2. for the seminar /lab activities	Laboratory with computers; C++ and programming language and Qt library

6.1. Specific competencies acquired [1]

[1] One can choose either competences or learning outcomes, or both. If only one option is chosen, the row related to the other option will be deleted, and the kept one will be numbered 6.

Professional/essential competencies	• Advanced programming skills in high-level programming languages (C/C++) • Development and maintenance of software systems
Transversal competencies	• Application of organized and efficient work rules, of responsible attitudes towards the didactic-scientific field, to bring creative value to own potential, with respect for professional ethics principles and norms • Use of efficient methods and techniques to learn, inform, research and develop the abilities to bring value to knowledge, to adapt at the requirements of a dynamical society and to communicate efficiently in Romanian language and in an international language

6.2. Learning outcomes

Knowledge	• The student knows to develop software programs and applications using object-oriented design and the C++ programming language. • The student has the ability to develop, design and create new applications using best practices of the field.
Skills	• The student has the necessary skills to understand and use object-oriented programming concepts to develop software applications of medium complexity. • The student is able to demonstrate the differences between traditional imperative design and object-oriented design, to explain class structures as fundamental, modular building blocks, to understand the role of inheritance, polymorphism, dynamic binding and generic structures in building reusable code. • The student has the ability to apply general rules to specific problems and produce relevant solutions.
Responsibility and autonomy:	• The student has the ability to work independently to write small/medium scale C++ programs using GUIs and to use classes written by other programmers when constructing their systems. • The student has the necessary skills to develop GUI applications using architectural templates suitable for specific user interaction applications.

7. Objectives of the discipline (outcome of the acquired competencies)

7.1 General objective of the discipline	• To prepare an object-oriented design of small/medium scale problems and to learn the C++ programming language, as well as to create graphical user interfaces using Qt.
7.2 Specific objective of the discipline	• To demonstrate the differences between traditional imperative design and object-oriented design. • To explain class structures as fundamental, modular building blocks. • To understand the role of inheritance, polymorphism, dynamic binding and generic structures in building reusable code. • To explain and to use defensive programming strategies, employing • formal assertions and exception handling. • To write small/medium scale C++ programs using Qt. • To use classes written by other programmers when constructing their • systems.

8. Content

8.1 Course	Teaching methods	Remarks
1. Basic elements in C • Basic elements of C/C++ language • Lexical elements. Operators. Conversions • Data types. Variables. Constants • Visibility scope and lifetime of the variables • C++ Statements • Function declaration and definition. Function overloading. Inline functions	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
2. Modular programming in C/C++ • Functions. Parameters • Pointers and memory management • Function pointers • Header files. Libraries • Modular implementations of ADTs	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
3. Object oriented programming in C++ • Classes and objects • Defining classes • Object creation and destruction • Operator overloading • Static and friend elements	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
4. Templates and the Standard Template Library • Function templates • Class templates • Containers, iterators in STL • STL algorithms	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
5. Inheritance • Simple inheritance and derived classes • Special functions in classes and inheritance • Substitution principle • Method overriding • Multiple inheritance • UML class diagrams and relations	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
6. Polymorphism • Inheritance, polymorphism • Static and dynamic binding • Virtual methods • Upcasting and downcasting • Abstract classes	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
7. Streams and exception handling • Input/Output streams • Insertion and extraction operators • Formatting. Manipulators. Flags • Text files • Exception handling. Exception-safe code	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
8. Resource management and RAII	• Interactive exposure	

• Resource Acquisition Is Initialization (RAII) • Smart pointers • RAII in STL. Smart pointers in STL	• Explanation • Conversation • Examples • Didactical demonstration	
9. Graphical User Interfaces (GUI) • Qt Toolkit: installation, Qt modules and instruments • Qt GUI components • Layout management • Qt Designer	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
10. Event driven programming elements • Callbacks • Events. Signals and slots in Qt • GUI design	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
11. Event driven programming elements • Model View Controller pattern • Models and Views in Qt • Using predefined models. Implementing custom models • Case study: Gene manager application	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
12. Design patterns • Creational, structural, behavioural patterns • Examples	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
13. Design patterns • Adapter pattern • Observer pattern • Iterator pattern • Composite pattern • Strategy pattern • Case study application and examples	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
14. Revision • Revision of the most important topics covered by the course • Examination guide	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	

Bibliography

1. B. Stroustrup. The C++ Programming Language, Addison Wesley, 1998.
2. Bruce Eckel. Thinking in C++, Prentice Hall, 1995.
3. A. Alexandrescu. Programarea moderna in C++: Programare generica si modele de proiectare aplicate, Editura Teora, 2002.
4. S. Meyers. Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition), Addison-Wesley, 2005.
5. S. Meyers. More effective C++: 35 New Ways to Improve Your Programs and Designs, Addison-Wesley, 1995.
6. B. Stroustrup. A Tour of C++, Addison Wesley, 2013.
7. C++ reference (http://en.cppreference.com/w/).
8. Qt Documentation (http://doc.qt.io/qt-6/).
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing, 1995.

8.2 Seminar	Teaching methods	Remarks
1. Simple problems in C. Functions.	• Interactive exposure	The seminar is structured as a 2

Structures and vectors. 2. Modular programming. 3. Classes. Operator overloading. User defined objects as class data members. Templates (dynamic vector). 4. Inheritance, polymorphism. 5. Files, exceptions. STL containers, iterators, algorithms. 6. Graphical User Interfaces 7. Complex problems. Implementation based on UML diagrams. Design patterns.	• Explanation • Conversation • Examples • Didactical demonstration	hour class, every 2 weeks.

Bibliography

1. B. Stroustrup. The C++ Programming Language, Addison Wesley, 1998.
2. Bruce Eckel. Thinking in C++, Prentice Hall, 1995.
3. A. Alexandrescu. Programarea moderna in C++: Programare generica si modele de proiectare aplicate, Editura Teora, 2002.
4. S. Meyers. Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition), Addison-Wesley, 2005.
5. S. Meyers. More effective C++: 35 New Ways to Improve Your Programs and Designs, Addison-Wesley, 1995.
6. B. Stroustrup. A Tour of C++, Addison Wesley, 2013.
7. C++ reference (http://en.cppreference.com/w/).
8. Qt Documentation (http://doc.qt.io/qt-6/).
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing, 1995.

8.3 Laboratory	Teaching methods	Remarks
1. Setting up a C++ compiler (MSVC/MinGW) and an IDE (Visual Studio). C/C++ general aspects. 2. Simple problems (in C). 3. Feature-driven software development process. Layered architecture. Test driven development. Modular programming. (I) 4. Feature-driven software development process. Layered architecture. Test driven development. Modular programming. (II) 5. Object oriented programming in C++. (I) 6. Object oriented programming in C++. (II) 7. Laboratory test. 8. Inheritance and polymorphism. 9. Text Files, exceptions. STL containers, iterators and algorithms. 10. Laboratory test. 11. Qt Graphical User Interfaces. (I) 12. Qt Graphical User Interfaces. (II) 13. Laboratory test. 14. Assignment delivery time.	• Explanation • Conversation	

Bibliography

1. B. Stroustrup. The C++ Programming Language, Addison Wesley, 1998.
2. Bruce Eckel. Thinking in C++, Prentice Hall, 1995.
3. A. Alexandrescu. Programarea moderna in C++: Programare generica si modele de proiectare aplicate, Editura Teora, 2002.
4. S. Meyers. Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition), Addison-Wesley, 2005.

5. S. Meyers. More effective C++: 35 New Ways to Improve Your Programs and Designs, Addison-Wesley, 1995.
6. B. Stroustrup. A Tour of C++, Addison Wesley, 2013.
7. C++ reference (http://en.cppreference.com/w/).
8. Qt Documentation (http://doc.qt.io/qt-6/).
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing, 1995.

9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program

- The course follows the ACM Curricula Recommendations for Computer Science studies.
- The course exists in the studying program of all major universities in Romania and abroad.
- The content of the course is considered by the software companies as important for average object oriented programming skills.

10. Evaluation

Activity type	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Percentage of final grade
10.4 Course	The correctness and completeness of the accumulated knowledge and the capacity to design and implement correct C++ programs.	Written examination (examination session)	30%
10.5 Seminar/laboratory	Be able to design, test and debug a C++ program with a graphical user interface.	Practical examination (examination session)	30%
	Correctness of delivered laboratory assignments and laboratory tests.	Program and documentation portfolio. Observation during the semester. Laboratory tests.	40%
10.6 Minimum standard of performance			

- Each student has to prove that they acquired an acceptable level of knowledge and understanding of the core concepts taught in the class, that they are capable of using knowledge in a coherent form, that they have the ability to establish certain connections and to use the knowledge in solving different problems in object oriented programming in C++.
- For participating at the examination attendance is compulsory for seminar and for laboratory activities, as follows: minimum 5 attendances for seminar and minimum 12 attendances for laboratory activities.
- Successfully passing of the examination is conditioned by a minimum grade of 5 for each of the following: laboratory activity, practical test and written examination.

11. Labels ODD (Sustainable Development Goals)[2]

[2] Keep only the labels that, according to the *Procedure for applying ODD labels in the academic process*, suit the discipline and delete the others, including the general one for *Sustainable Development* – if not applicable. If no label describes the discipline, delete them all and write „*Not applicable*.”.

Not applicable.

Date: 15.04.2025	Signature of course coordinator Assoc. Prof. PhD. Bocicor Maria Iuliana	Signature of seminar coordinator Assoc. Prof. PhD. Bocicor Maria Iuliana

Date of approval: ...	Signature of the head of department Assoc.prof.phd. Adrian STERCA