SYLLABUS

Object Oriented Programming

University year 2025-2026

1. Information regarding the programme

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

2. Information regarding the discipline

2.1. Name of the discipline	**Object oriented programming basics**					Discipline code	**MLE5234**
2.2. Course coordinator				**Lect. PhD Diana Laura Borza**			
2.3. Seminar coordinator				**Lect. PhD Diana Laura Borza**			
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	**42**

Time allotment for individual study (ID) and self-study activities (SA)	**hours**
Learning using manual, course support, bibliography, course notes (SA)	24
Additional documentation (in libraries, on electronic platforms, field documentation)	15
Preparation for seminars/labs, homework, papers, portfolios and essays	19
Tutorship	9
Evaluations	13
Other activities:	

3.7. Total individual study hours	**80**
3.8. Total hours per semester	**150**
3.9. Number of ECTS credits	**6**

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, having a C++ compiler, a C++ IDE (preferably Visual Studio) and Qt library installed

6. 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	● mathematical processing of data, analysis and interpretation of some phenomena and processes ● development and analysis of algorithms for solving problems ● to design mathematical models describing some real phenomenon ● programming in high level languages ● analysis, testing and using of software system
Transversal competencies	● application of rigorous and efficient work rules, manifestation of responsible attitudes towards the didactic-scientific field, to bring optimal and creative values to own potential in specific situations, with respect to professional ethics principles and norms ● efficient and effective development of organized activities of teamworks ● use of efficient information resources and techniques to learn and develop the professional abilities in Romanian language and in an international language

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

7.1 General objective of the discipline	● To understand the concepts of the object-oriented programming paradigm and to design object-oriented solutions of small/medium scale problems, using C++ and 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 design user- interfaces and write small/medium scale C++ programs using Qt. ● To use classes written by other programmers and third-party libraries when constructing their systems.

8. Content

8.1 Course	Teaching methods	Remarks
1. **C/C++ introduction** (basic elements of C/C++ programming language, data types, constant variables, scope and lifetime of the variables, statements, functions: declaration and definition, overloading functions).	● Interactive exposure ● Explanation ● Conversation ● Examples ● Didactical demonstration	
2. **Modular programming in C/C++** (functions, formal and actual parameters, pointers and memory management, the stack and the help, pointers to functions, header files, modular programming, libraries).	● Interactive exposure ● Explanation ● Conversation ● Examples ● Didactical demonstration	
3. **Object oriented programming in C++** (introduction to object oriented programming, object oriented programming features, abstraction, encapsulation, classes and objects, access modifiers, object creation and destruction, operator overloading, static and friend elements).	● Interactive exposure ● Explanation ● Conversation ● Examples ● Didactical demonstration	
4. **Inheritance and polymorphism** (base and derived classes, Liskov substitution principle, method overriding, inheritance and polymorphism).	● Interactive exposure ● Explanation ● Conversation ● Examples ● Didactical demonstration	
5. **Polymorphism** (static and dynamic binding, virtual methods, multiple inheritance,	● Interactive exposure ● Explanation ● Conversation	

upcasting and downcasting, abstract classes, UML class diagrams and relations).	• Examples • Didactical demonstration	
6. **Templates in C++. The C++ Standard Template Library** (function templates, class templates, containers in STL: array, vector, list, stack, heap, map, set), iterators, STL algorithms, lambda functions.	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
7. **Streams and exception handling** (input output streams, insertion and extraction operators, overloading 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** (Resource Acquisition Is Initialization (RAII), smart pointers, move semantics, smart pointers in STL: std::unique_ptr, std::shared_ptr, std::weak_ptr)	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
9. **Graphical User Interfaces** (Qt Toolkit: installation, Qt modules and instruments, Qt GUI components, Layout management, design interfaces using Qt Designer).	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
10. **Event driven programming I** (callbacks, events, signals and slots in Qt).	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
11. **Event driven programming II** (Model View Controller, Models and Views in Qt, using predefined models, implementing custom models).	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
12. **Design patterns I** (creational, structural, behavioral patterns, examples, singleton, factory method, adapter pattern).	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
13. **Design patterns II** (façade pattern, observer 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 modernă în C++: Programare generică și 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-5/).
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 / laboratory	Teaching methods	Remarks
Seminar		
1. Simple problems in C. Functions. Structures, enums and arrays.	• Interactive exposure • Explanation • Conversation	The seminar is structured as a 2 hour class, every 2 weeks.
2. Modular programming.	• Interactive exposure • Explanation • Conversation	
3. Classes. Operator overloading. User-defined objects as class data members.	• Interactive exposure • Explanation • Conversation	
4. Inheritance. Polymorphism. Templates.	• Interactive exposure • Explanation • Conversation	
5. Files, exceptions. STL containers, iterators, algorithms.	• Interactive exposure • Explanation • Conversation	
6. Graphical User Interfaces.	• Interactive exposure • Explanation • Conversation	
7. Implementation based on UML diagrams. Design patterns.	• Interactive exposure • 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 modernă în C++: Programare generică și 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-5/).
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. ***Design Patterns: Elements of Reusable Object-Oriented Software***, Addison-Wesley Longman Publishing, 1995.

Laboratory		
1. Environment setup (installing a C++ compiler and an IDE). C/C++ basics.	• Explanation • Conversation	The laboratory is structured as weekly 2 hour classes.
2. Introductory problems (in C).	• Explanation • Conversation	
3. Feature-driven software development process. Layered architecture. Test driven development. Modular programming	• Explanation • Conversation	
4. Classes and objects in C++. Copy constructors, assignment operators, destructors.	• Explanation • Conversation	
5. Inheritance. Method overriding.	• Explanation • Conversation	
6. Inheritance and polymorphism. Virtual methods.	• Explanation • Conversation	
7. Laboratory test.	Practical test	
8. STL containers, iterators and algorithms.	• Explanation • Conversation	
9. Streams, overloading the insertion and extraction operators, persistence.	• Explanation • Conversation	

10. Exception handling. Testing.	• Explanation • Conversation	
11. Qt Graphical User Interfaces I.	• Explanation • Conversation	
12. Qt Graphical User Interfaces II. Signals and slots in Qt.	• Explanation • Conversation	
13. Design patterns.	• Explanation • Conversation	
14. Laboratory test.	Practical test	

Bibliography

1. B. Stroustrup. ***The C++ Programming Language***, Addison Wesley, 1998.
2. R. Gilberg. **C++ Programming: An Object-Oriented Approach**, McGraw-Hill Education, 2019
3. A. Alexandrescu. ***Programarea modernă în C++: Programare generică și 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.
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-5/).
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. ***Design Patterns: Elements of Reusable Object-Oriented Software***, Addison-Wesley Longman Publishing, 1995.
10. Bruce Eckel. ***Thinking in C++***, Prentice Hall, 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 respects 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 (regular session).	**60%**
10.5 Seminar/laboratory	Ability to design, implement, test and debug a C++ program with a graphical user interface.	Practical evaluation. Two tests during the semester.	**20%**
	Project.	Design, implementation and testing of a small-medium application that uses a 3-tier architecture. Documentation	**20%**

10.6 Minimum standard of performance

- Students must 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 this knowledge in a coherent form, that they have the ability to establish certain connections and to use the knowledge in solving small/medium scale problems using object-oriented programming in C++.
- Successfully passing the examination is conditioned by a minimum grade of 5 (no rounding) for the laboratory practical test, the laboratory assignment and written examination.
- Attendance is mandatory for 5 seminar sessions and 12 laboratory sessions.

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

Not applicable.

Date:	Signature of course coordinator	Signature of seminar coordinator
April 27, 2025.	Lect. PhD. Diana Laura Borza	Lect. PhD. Diana Laura Borza

Date of approval:
...

Signature of the head of department

Assoc.prof.phd. Adrian STERCA

[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.*".