COURSE UNIT DESCRIPTION

Course unit title	Course unit code
Object-Oriented Programming in C++	

Annotation
The course covers essential programming concepts and principles that encourage abstraction-based thinking and implements the principle of information hiding, thus facilitating code quality, readability, ease of maintenance and ease of change. During the course, student learns various tools and technologies, and studies object-oriented design patterns that allow to construct standard solutions to typical problems. To take the course, students are expected to already have the fundamentals of programming – the course can be viewed as a natural extension of a course on Procedural Programming Fundamentals taught in previous semester.

Lecturer(s)	Department where the course unit is delivered
Coordinator: Irmantas Radavičius **Other lecturers:** Viktoras Golubevas	Faculty of Mathematics and Informatics Vilnius University

Cycle	Type of the course unit
First	Optional, individual studies

Mode of delivery	Semester or period when the course unit is delivered	Language of instruction
Mixed	2nd semester	Lithuanian, English

Prerequisites	
Prerequisites: Procedural programming	**Other requirements:** none

Number of credits allocated	Workload of the student	Contact hours	Individual work
5	134	66	68

Purpose of the course unit: program competences to be developed
The purpose of the course unit: • To familiarize oneself with the C++ programming language, STL and other C++ libraries • To familiarize oneself with object-oriented and generic programming paradigms • To ground oneself in fundamentals of object-oriented analysis and design, and learn basics of UML language Generic competences: • will be able to organise their own work independently (GK 1.3) • will recognize of the need for, and engage in life-long learning (GK 2.1) • will be able to independently acquire new knowledge, methods and tools and apply them in practice. (GK 2.3) • will understand professional and ethical responsibility (GK 3.1) Specific competences: • will be able to apply mathematical foundations, knowledge of science and engineering, computer science theory, and algorithmic principles in software systems development (SK 4.2) • will be able to reason at abstract level, to use formal notation, to prove the correctness, and to apply formalisation and specification for real-world problems (SK 4.3) • will be able to use existing hardware, software and application systems, to identify, understand and apply the promising technologies (SK 6.3) • will be aware of project management, quality assurance, and process improvement practices and develop abilities to apply them (SK 6.6.)

Learning outcomes of the course unit: students will be able to	Teaching and learning methods	Assessment methods
• apply object-oriented techniques to model real world situations • understand, modify and create source code in C++ programming language • create, test, and document C++ applications	Lectures Assignments Individual work	Projects (individually and in groups) Exam (written)

Course content: breakdown of the topics	Contact hours						Individual work: time and assignments	
	Lectures	Tutorials	Seminars	Practice	Laboratory work	**Contact hours**	**Individual work**	**Assignments**
Course overview. Introduction. Object-oriented programming. Object-oriented programming languages. C++ history. C++ as a "better C".	2				2	**4**	**4**	Individual reading Projects (individually and in groups)
Classes and Objects. Fields and Methods. Access control. Object lifecycle. Constructors and Destructors. Static elements.	2				2	**4**	**4**	
Multiple file applications. Namespaces. Header files. Error correction and prevention. Unit tests. Intermediate output. Error handling. Assertions. Exception handling.	2				2	**4**	**4**	
Methods, Method overloading. Constants, Constant fields and methods. Parameter passing. Pointers and References. Operator overloading. Friend classes and functions.	2				2	**4**	**4**	
Composition. Types of composition. Object shallow and deep copy. Inner classes. Arrays. Standard containers.	2				2	**4**	**4**	
Object-oriented analysis and design. UML language. UML use case, activity, class, and sequence diagrams.	2				2	**4**	**4**	
Generic programming. Function and class templates.	2				2	**4**	**4**	
STL containers. STL iterators.	2				2	**4**	**4**	
STL functional objects. STL algorithms.	2				2	**4**	**4**	
Inheritance. Method overriding. Virtual methods. Polymorphism. Typecasting.	2				2	**4**	**4**	
Types of inheritance. Access control. "Diamond problem". Virtual inheritance.	2				2	**4**	**4**	
Abstract classes and related design patterns.	2				2	**4**	**4**	
Object creation and copying, and related design patterns.	2				2	**4**	**4**	
Revisiting object-oriented analysis and design, remaining important design patterns.	2				2	**4**	**4**	
Resource management. Exception handling and safety.	2				2	**4**	**4**	
Preparation for the exam.	2				2	**4**	**8**	
Exam (written).						**2**		
Total:	32				32	**66**	**68**	

Assessment strategy	Weight, perc.	Deadline	Assessment criteria
Projects (individually and in groups)	60	During the semester students get assignments that must be submitted following the order announced in the beginning of the semester	During the semester student is expected to participate in two projects, each of which is assessed in three stages. Accordingly, the total number of assignments is 6 (six), and each of them has an equal weight (10%). First project is an individual programming assignment, to create a package of classes to model a specific real-world situation (assessed in three stages, based on adhering to the deadlines and requirements for each stage). Second project can be a team project (in that case, the assessment considers the degree of the contribution for each member), for the final stage students must submit an application, built according to the principles of object-oriented programming, analysis, and design. To be allowed to take an exam, students must submit (and successfully pass) at least 3 (three) assignments. The lecturer can give up to 1 bonus extra point, based on the effort and results demonstrated by the student.
Exam (written)	40	June	During the exam, students solve problems of various types and difficulty. Exam consists of two parts: the test (students answer questions of various types) and writing the code (students must submit a working code satisfying the requirements provided).
Extern		The student can repeat the course externally, if before they have participated fully and they accept the previously collected number of points. In this case, the points get accounted for and the student only repeats the exam. The student who is taking the course unit externally must inform the lecturer in the beginning of the semester and get the written consent with the above-mentioned number of points confirmed. If the student has not collected the minimal number of points required to pass, or the number of points collected does not suit the student, the subject cannot be repeated externally.	

Author	Publishing year	Title	Number or volume	Publisher or URL
Required reading				
Bjarne Stroustrup	2013	The C++ Programming Language	4th ed.	Addison-Wesley
Paul J. Deitel, Harvey M. Deitel	2016	C++ How to Program	10th ed.	Pearson
Recommended reading				
Bruce Eckel	2000	Thinking in C++	2nd ed.	https://archive.org/details/TICPP2ndEdVolOne
Bruce Eckel	2003	Thinking in C++, Volume Two: Practical Programming	1st ed.	https://archive.org/details/TICPP2ndEdVolTwo
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides	1994	Design Patterns: Elements of Reusable Object-Oriented Software	1st ed.	Addison-Wesley