Course Title	**Programming Principles II**				
Course Code	**ACSC183**				
Course Type	**Compulsory**				
Level	**BSc (Level 1)**				
Year / Semester	**1st , 2nd**				
Teacher's Name	**Andreas Constantinides, PhD**				
ECTS	5	Lectures / week	2	Laboratories/week	2
Course Purpose	This course aims to introduce students to more advance programming principles using the C++ programming language with emphasis on good programming practices, algorithmic design, error handling and effective programming with the usage of libraries (such as the c-string, cctype, cstdlib) and implementation of header files, I/O streams, the string class, pointers, structures and classes. Major focus is given on the introduction to students to Object-oriented programming principles and applications.				
Learning Outcomes	By the end of the course, students should be able to: • Recognize the use of structures and arrays and apply them to construct and manipulate programmatically composite data types. • Develop programs that read and write from/to files and handle strings. • Show the ability to use pointers and dynamic data types in programming exercises. • Design medium size programs using a modular approach and apply the use of functions for developing algorithmic units and their communication through parameter passing and function returns. • Examine the benefits and core concepts of object-oriented C++ programming, and develop simple programs using object-oriented terminology.				
Prerequisites	ACSC182	Corequisites			
Course Content	This course consists of the following chapters: • **Review of Programming Principles:** Data and Algorithms. Variables, operators and statements. Sequential, Selective and iterative processes. Understanding complex code. Interacting with input and output. • **Using Composite Data Types:** Limitations of built-in data types. Mechanisms for constructing composite data types: Arrays and Structures. Building complex data types. Combining arrays and structures. Handling composite types. Efficiency considerations. Composite data types in functions. • **Pointers and Dynamic Data:** Understanding pointers and memory addresses. Pointer syntax. Parameter Passing revisited.				

	• **Pointers in Programmatic Development**: Pointers and arrays. Dynamic memory allocation. Memory allocation lifetime and memory leaks. String handling. • **Persistent Storage:** Files, file pointers and file objects. Reading and writing methods. File manipulation. • **Modular Programming Revisited:** Tackling larger programming tasks. Dividing the problem into parts. Designing software solutions. Effects of structures, pointers and dynamic content in functions. Breaking programs into units. Use of header files and custom-made libraries. • **Combining Data and Algorithms – Introduction to Object-Orientation:** Introducing a new way of programming: functions as members of 'objects'. Key features of object-orientation: Classes, objects, properties and methods. Objects and memory management. • **Laboratory Work:** The role of the C++ programming language as a tool for solving advanced mathematical and engineering problems is emphasised through practical work carried out.
Teaching Methodology	The course is structured around lectures (2 hours per week) and laboratories (2 housr per week) as well as homeworks/assignemts, laboratory exercises, practical exercises, demonstrations and individual work. During the lectures, students are encouraged to participate in discussions enabling the exchange of ideas and examples. Laboratory exercises are handed to students and their solutions are discussed at laboratory periods. Additional tutorial time at the end of each lecture is provided to students as well as additional notes for each section of the course and worksheets, which process in the lab or as homework. Students are expected to demonstrate the necessary effort to become confident with the different concepts and topics of the course. Lecture notes and presentations are available through the web (e-learning platform) for students to use in combination with the textbooks. Furthermore, theoretical principles are explained by means of specific examples and for solving specific problems using practical examples. Students are also advised to use the subject's textbook or reference books for further reading and practice.
Bibliography	Textbooks: • Walter Savitch, Problem Solving with C++, Addison-Wesley, 7th Ed., 2009, ISBN: 0-321-54940-6.References: References: • Harvey M. Deitel, Paul J. Deitel, C++ How to Program, Prentice Hall, 7th Ed., 2010, ISBN: 0-136-11726-0.
Assessment	The Students are assessed via continuous assessment throughout the duration of the Semester, which forms the Coursework grade and the final written exam. The coursework and the final exam grades are weighted 40% and 60%, respectively, and compose the final grade of the course. Various approaches are used for the continuous assessment of the students, such as mid-term test, class participation and laboratory work,

	homework and assignments. The assessment weight, date and time of each type of continuous assessment is being set at the beginning of the semester via the course outline. An indicative weighted continuous assessment of the course is shown below: • **Mid-term Test** (10% of total marks for module) • **Participation Activities (Lab work)**(10% of total marks for module) • **Homework/Assignment** (10% of total marks for module) • **Lab Tests** (10% of total marks for module) • **One closed-book, 3-hours exam** (60% of total marks for module Students are prepared for final exam, by revision on the matter taught, problem solving and concept testing. The final assessment of the students is formative and summative and is assured to comply with the subject's expected learning outcomes and the quality of the course.
Language	**English**