MODULE DESCRIPTION FORM

Module Information					
Module Title	**Programming Fundamentals II**			**Module Delivery**	
Module Type	**Core**			☒ **Lecture** ☒ **Practical**	
Module Code	**IT203**				
ECTS Credits	**7**				
SWL (hr/sem)	**175**				
Module Level	UGI		**Semester of Delivery**	2	
Administering Department	Information Technology		College	Science	
Module Leader	Ali Mahmoud Ali		**e-mail**	ali.mahmoud@uowa.edu.iq	
Module Leader's Acad. Title	Assistant Lect.		**Module Leader's Qualification**	Mc.S	
Module Tutor	Ali Mahmoud Ali		**e-mail**	ali.mahmoud@uowa.edu.iq	
Peer Reviewer Name	Assist. Prof. Dr. Haider Mohammed		**e-mail**	hayder.alghanami@uowa.edu.iq	
Scientific Committee Approval Date	2026-2025		**Version Number**	V1	

Relation with other Modules			
Prerequisite module	**Programming Fundamentals 1**	**Semester**	1
Co-requisites module	-	**Semester**	…

Department Head Approval **Dean of the College Approval**

Module Aims, Learning Outcomes and Indicative Contents	
Module Objectives	The following are some key aims and benefits of studying Programming Fundamentals II: 1. Problem Solving: Teach students how to analyze problems and develop algorithms to solve them. Emphasize problem-solving techniques, algorithm design, and decomposition of complex problems into smaller, manageable parts. 2. Debugging and Testing: Teach students how to debug and test their programs to identify and fix errors. Explore techniques for error detection, debugging tools, and strategies for writing effective test cases. 3. Data Structures: Introduce students to fundamental data structures such as arrays, stacks, queues, structures. Explore their properties, implementation, and usage in solving programming problems. 4. Modular Programming: Introduce the concept of modular programming, including the use of functions, parameter passing, and code reuse. Emphasize the importance of modular design and writing reusable and maintainable code. 5. Programming Best Practices: Introduce students to programming best practices and coding standards, including code documentation, naming conventions, code formatting, and code optimization techniques. 6. Files Input and Output: Teach students how to interact with the user and handle input/output operations, including reading from and writing to files, standard input/output, and error handling. 7. Introduction to Object-Oriented Programming (OOP): Introduce the principles and concepts of OOP, including classes.

Module Learning Outcomes	The following are some common learning outcomes for an Programming Fundamentals II : 1. Effective Code Writing: Write clear, well-structured, and readable code that follows coding standards and best practices, including proper indentation, meaningful variable names, and appropriate comments. 2. Use of Data Structures: Apply appropriate data structures, such as arrays, linked lists, stacks, and queues, to store and manipulate data effectively in programming problems. 3. Modular Design and Reusability: Design and implement modular programs by breaking them into reusable functions or methods, facilitating code reuse, improving maintainability, and promoting good software engineering practices. 4. Debugging and Testing Skills: Use debugging techniques and tools to identify and fix errors in programs. Develop effective test cases and perform testing to ensure program correctness and reliability. 5. Understanding of Object-Oriented Programming (OOP) Concepts.
Indicative Contents	The indicative contents of an Programming Fundamentals II module have a list of common topics that shown below : 1- Modular Programming: **[25 hrs]** Functions and procedures, Scope and lifetime of variables, Parameter passing mechanisms. 2- Data Structures: **[25 hrs]** Arrays, Strings and lists, Structures, Stacks and queues. 3- Input and Output: **[15 hrs]** Standard input/output, Reading from and writing to files, Error handling and exception handling. 4- Debugging and Testing: Common types of programming errors, Debugging techniques and tools. **[20 hrs]** 5- Object-Oriented Programming (OOP) Concepts: Classes and objects. **[5 hrs]**

Learning and Teaching Strategies	
Strategies	To teaching an Programming Fundamentals II module, various strategies can be employed to facilitate effective learning and engagement. Here are some learning and teaching strategies commonly used in Programming Fundamentals II module: 1- Lectures: Delivering lectures to present theoretical concepts, principles, and foundational knowledge of Programming Fundamentals II. Lectures can include visual aids, examples, and demonstrations to enhance understanding. 2- Interactive Discussions: Encourage students to actively participate in discussions by asking questions, sharing their thoughts, and engaging in peer-to-peer learning. Discussions can focus on challenging concepts, real-world applications, or case studies related to Programming Fundamentals II. 3- Hands-on Lab Sessions: Conduct practical lab sessions where students can gain hands-on experience with Programming Fundamentals II, commands, and programming exercises. These sessions provide an opportunity to reinforce theoretical concepts and develop practical skills. 4- Group Projects: Assign group projects that involve designing, implementing, and evaluating components of an Programming Fundamentals II. Group projects promote teamwork, problem-solving, and practical application of operating system concepts. 5- Online Resources and Tutorials: Provide access to online resources, tutorials, and interactive learning materials related to Programming Fundamentals II. This allows students to explore additional content, reinforce their understanding, and self-assess their progress. 6- Assessments and Feedback: Use a variety of assessment methods such as quizzes, assignments, projects, and exams to evaluate students' understanding of Programming Fundamentals II concepts. Provide timely and constructive feedback to help students improve their knowledge and skills.

Student Workload (SWL)

Structured SWL (h/sem)	75	**Structured SWL (h/w)**	5
Unstructured SWL (h/sem)	97	**Unstructured SWL (h/w)**	6.5
Total SWL (h/sem)	172 + 3 final = 175		

Module Evaluation

		Time/Number	Weight (Marks)	Week Due	Relevant Learning Outcome
Formative assessment	**Quizzes**	5	1% (5)	All Weeks	1,2,3,4
	Assignments	5	1%(5)	All Weeks	All Outcome
	Lab	5	4% (20)	All Weeks	All Outcome
	Home Work	5	2%(10)	All Weeks	All Outcome
Summative assessment	**Midterm Exam**	2hr	10%(10)	9	
	Final Exam	3hr	50% (50)	17	
Total assessment			100% (100 Marks)		

Delivery Plan (Weekly Syllabus)

	Material Covered
Week 1	Functions
Week 2	Function Types
Week 3	The concept of Recursion
Week 4	Array
Week 5	1D array

Week 6	2D array (Matrix)
Week 7	Array of Characters (Strings)
Week 8	String Processing
Week 9	Midterm Exam
Week 10	Arrays and functions
Week 11	Structures
Week 12	Array of structures and Nested Structures
Week 13	Stack and Queue
Week 14	Pointers
Week 15	Files
Week 16	**Preparatory week before the final Exam**

Delivery Plan (Weekly Lab. Syllabus)	
	Material Covered
Week 1	Writing Codes using Functions
Week 2	Writing Codes using Function Types
Week 3	Writing Codes using The concept of Recursion
Week 4	Writing Codes using Arrays
Week 5	Writing Codes using 1D arrays
Week 6	Writing Codes using 2D array (Matrix)s
Week 7	Writing Codes using Array of Characters (Strings)
Week 8	Writing Codes using String Processing
Week 9	Midterm Exam
Week 10	Writing Codes using Arrays and functions
Week 11	Writing Codes using Structures
Week 12	Writing Codes using Array of structures and Nested Structures
Week 13	Writing Codes using Stack and Queue

Week 14	Pointers
Week 15	Files

Learning and Teaching Resources

	Text	Available in the Library?
Required Texts	**C++: The Complete Reference, Fourth Edition, Herbert Schildt.**	**Yes**
Recommended Texts	**The C++ Programming Language , Third Edition , Bjarne Stroustrup.**	**Yes**
Websites	**https://stackoverflow.com/**	

Grading Scheme

Group	Grade	Mark	Marks %	Definition
Success Group (50 - 100)	**A -** Excellent	Excellent	90 - 100	Outstanding Performance
	B - Very Good	Very Good	80 - 89	Above average with some errors
	C - Good	Good	70 - 79	Sound work with notable errors
	D - Satisfactory	Fair / Average	60 - 69	Fair but with major shortcomings
	E - Sufficient	Pass / Acceptable	50 - 59	Work meets minimum criteria
Fail Group (0 – 49)	**FX –** Fail	Fail (Pending)	(45-49)	More work required but credit awarded
	F – Fail	Fail	(0-44)	Considerable amount of work required

Note: Marks Decimal places above or below 0.5 will be rounded to the higher or lower full mark (for example a mark of 54.5 will be rounded to 55, whereas a mark of 54.4 will be rounded to 54. The University has a policy NOT to condone "near-pass fails" so the only adjustment to marks awarded by the original marker(s) will be the automatic rounding outlined above.