MODULE DESCRIPTION FORM

Module Information					
Module Title	**Object-Oriented Programming II**			**Module Delivery**	
Module Type	**Core**			☒ **Lecture**	
Module Code	**IT2202**			☒ **Practical**	
ECTS Credits	**6**				
SWL (hr/sem)	**150**				
Module Level	UG2		**Semester of Delivery**	2	
Administering Department	Information Technology		**College**	College Science	
Module Leader	Mohsin Hassan Hussein		**e-mail**	mohsin.ha@uowa.edu.iq	
Module Leader's Acad. Title	Asst. Professor		**Module Leader's Qualification**	Ph.D.	
Module Tutor	Mohsen Hassan Hussein		**e-mail**	mohsin.ha@uowa.edu.iq	
Peer Reviewer Name	Asst. Prof Haider Mohammed		**e-mail**	hayder.alghanami@uowa.edu.iq	
Scientific Committee Approval Date	2025-01-20		**Version Number**	1.0	

Relation with other Modules			
Pre-requisite module	**Object-Oriented Programming1**	**Semester**	1
Co-requisites module	-	**Semester**	

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

Module Aims, Learning Outcomes and Indicative Contents

Module Aims

1. Provide a sound knowledge of the underlying principles and experience in the practical application of this course is essential for any information technology specialist.
2. extend students with procedural programming knowledge and skills in the object-oriented paradigm and builds experience with interpreted languages to introduce compiled languages.
3. In addition to further shaping a solid development methodology, the course prepares students for continued investigation into advanced programming topics.
4. develop a wide range of software solutions for real-world scenarios.

Module Learning Outcomes

On completion of this course students will be able to:

1. Compare and contrast interpreted vs compiled languages; and prototype-based vs class-based languages;
2. Competently apply the concepts of polymorphism, inheritance, encapsulation, exception handling, memory management, threads, and file I/O;
3. Design, code, verify, test, document, amend and refactor moderately complex programs meeting requirements by applying object-oriented principles;
4. Contribute to reviews of own work with others through the use of collaborative tools.
5. develop programs using the C++ Standard for real-world.

Indicative Contents

Topics

	Description	Weighting (75%)
1.	Compiled languages; imperative programming versus object orientation	5.00
2.	Objects and classes	10.00
3.	Inheritance	15.00
4.	Polymorphism	15.00
5.	Templates functions and classes	15.00
6.	Exception handling	15.00

Learning and Teaching Strategies

Strategies

Overview Strategies

Object-oriented programming is one of the principle paradigms in software development used by organisations worldwide to develop a wide range of software solutions. Sound knowledge of the underlying principles and experience in the practical application of these is essential for any information technology specialist. This intermediate programming course extends students' procedural programming knowledge and skills into the object-oriented paradigm and builds on previous experience with interpreted languages to introduce compiled languages. In addition to further shaping a solid development methodology, the course prepares students for continued investigation into advanced programming topics.

This course extends the student's basic procedural design and programming knowledge into the object-oriented paradigm. The student will be expected to learn and apply the basic concepts of object-oriented design and programming, i.e., abstraction, inheritance, and polymorphism, in the context of the C++ language through giving lectures, practical exercises within the laboratories, assignments about some specific topics, and small projects.. Key software engineering principles such as decomposition and component re-use will also be emphasised.

Student Workload (SWL)			
Structured SWL (h/sem)	75	**Structured SWL (h/w)**	5
Unstructured SWL (h/sem)	72	**Unstructured SWL (h/w)**	4.8
Student workload expectations (SWL &USWL) To do well in this subject, students are expected to commit approximately 10 hours per week including class contact hours, independent study, and all assessment tasks. If you are undertaking additional activities, the weekly workload hours may vary.			
Total SWL (h/sem)	147 + 3 final = 150		

Module Evaluation					
		Time/ Number	**Weight (Marks)**	**Week Due**	**Relevant Learning Outcome**
Formative assessment	**Quizzes**	5	10% (8)	All Weeks	1,2,3,4,5
	Onsite Assignments	5	10% (5)	All Weeks	1,2,3,4,5
	Home Work	5	10% (7)	All Weeks	1,2,3,4,5
	Project	1	10% (5)	All Weeks	1,2,3,4,5
	Labs	5	10% (15)	All Weeks	1,2,3,4,5
Summative assessment	Midterm Exam	2hr	10% (10)	7	
	Final Exam	3hr	50% (50)	16	
Total assessment			100% (100 Marks)		

Delivery Plan (Weekly Syllabus)		
	Material Covered	Weighting (30+5=35%)
Week 1	- The fundamental concepts of programming, including procedural and object-oriented programming will be introduced. Also, consider the basic principles behind object-oriented programming techniques, including objects, classes, inheritance, and polymorphism. Then you will get started in programming environment by applying what you have learned.	2
Week 2	- Introduction about objects and classes, class declaration, Object declaration, with examples.	2
Week 3	**Inheritance** - Base-Class Access Control - Inheritance and protected Members - Protected Base-Class Inheritance - Inheriting Multiple Base Classes	2
Week 4	**Inheritance …cont'd** - Constructors, Destructors, and Inheritance - When Constructor and Destructor Functions Are Executed - Passing Parameters to Base-Class Constructors	2
Week 5	**Polymorphism** - Virtual Functions - A pointer of base class type - Virtual Base Classes - Calling a Virtual Function Through a Base Class Reference	2
Week 6	**Polymorphism …cont'd** - The Virtual Attribute Is Inherited - Virtual Functions Are Hierarchical - Pure Virtual Functions - Abstract Classes	2
Week 7	**Mid-Term Exam Revision**	2
Week 8	**Templates** - Generic Function - A Simple Function Template - A function with two generic types - What the Compiler Does	2
Week 9	**Templates … cont'd** - Overloading a Function Template - Using Standard Parameters with Template Functions - Template Arguments Must Match	2
Week 10	**Templates …cont'd** - Function Templates with Multiple Arguments - Template Arguments Must Match - Syntax Variation - Class Templates - An Example with Two Generic Data Types	2
Week 11	**Handling Exceptions** - Exceptions - Why Do We Need Exceptions?	2

	- Exceptions Syntax - Exception Mechanism - Throwing Exceptions - Catch Base and Derived classes with Exceptions	
Week 12	**Handling Exceptions…cont'd** - Constructor and Destructor with Exceptions - Re-thrown Exceptions - Nested Exceptions - Handling Exceptions Class Activities	2
Week 13	**Handling Exceptions…cont'd** - Handling Exceptions inside Function - Catching All Exception - Restricting Exceptions	2
Week 14	**Handling Exceptions…cont'd** - Using Threw()-To Restric any types of Exceptions - Re-thrown Exception inside function - Handling Exceptions Class Activities	2
Week 15	- Students course workload evaluation.	2
Week 16	**Prepare to the final Exam**	3

Delivery Plan (Weekly Lab. Syllabus)		
	Material Covered	**Weighting (45%)**
Week 1 - Lab 1	- Prepare OOP environment, overview about unified modeling language (UML) diagram. - Access to a standard C++ or Python compiler - Linux g++ compiler and its equivalent MinGW running under windows.	3
Week 2 - Lab 2	**Introduction to Classes and Objects** - Understand function call by value method of parameter passing - Understand Pass parameters by reference method	3
Week 3 - Lab 3	**Apply Inheritance concept using many programming codes include:** - Base-Class Access Control - Inheritance and protected Members - Protected Base-Class Inheritance - Inheriting Multiple Base Classes	3
Week 4 – Lab 4	- Implement inheritance concept using Constructors, Destructors Functions and learn how to Pass Parameters to Base-Class Constructors.	3
Week 5 – Lab 5	**Implement the Polymorphism concept using many C++ code examples which include:** - Implement Virtual Functions - Use A pointer of base class type - Implement Virtual Base Classes - Calling a Virtual Function Through a Base Class Reference	3
Week 6 – Lab 6	- Implement the Virtual Attribute and Virtual Functions	3

	- Implement Pure Virtual Functions - Implement Abstract Classes	
Week 7 – Lab 7	- Labs exam1 with evaluation	3
Week 8 – Lab 8	**Implemented the Template concept using many examples code in C++:** - Understand the Generic Function - Implement A Simple Function Template - Implement function template with two generic types	3
Week 9 – Lab 9	- Implement Overloading a Function Template - Implement Using Standard Parameters with Template Functions - Prove the Template Arguments Must be Match	3
Week 10 – Lab 10	**Code examples to implement:** - Function Templates with Multiple Arguments - Class Templates - Template with Two Generic Data Types	3
Week 11 – Lab 11	**Implemented the Handling Exception concept** - Basic code exception - How can Throw Exceptions - Catch Base and Derived classes with Exceptions examples	3
Week 12 – Lab 12	**Code examples to implement:** - Constructor and Destructor with Exceptions - Re-thrown Exceptions - Nested Exceptions	3
Week 13 – Lab 13	- Labs exam2 with evaluation	3
Week 14 – Lab 14	- Code example to implement Exceptions inside Function - How to Catch All Exception? - How can Restrict Exceptions?	3
Week 15 – Lab 15	- OOP II project Implementation with discussion for each student	3

Learning and Teaching Resources		
	Text	**Available in the Library?**
Required Texts	1. Malik, D.S 2018, *C++ Programming: Program Design Including Data Structures*, 8th edn, Cengage. (ISBN 978-1-337-11756-2.) 2. OOP – Learn Object Oriented Thinking and Programming, ISBN-10: 8090466184, Tomas Bruckner, 2013. 3. The student must have access to a standard C++ compiler. The only supported compilers are the Linux g++ compiler and its equivalent MinGW running under Windows.	No

Recommended Texts	4. Object-Oriented Programming Using C++ Fourth Edition by Joyce Farrell	No
Websites		

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.