MODULE DESCRIPTION FORM

نموذج وصف المادة الدراسية

Module Information معلومات المادة الدراسية					
Module Title	**Computer Fundamentals and Programming I**			**Module Delivery**	
Module Type	**Basic**			☒ **Theory** ☐ **Lecture** ☒ **Lab** ☒ **Tutorial** ☐ **Practical** ☐ **Seminar**	
Module Code	**UREQ111**				
ECTS Credits	**4**				
SWL (hr/sem)	**100**				
Module Level	1		**Semester of Delivery**	1 (one)	
Administering Department			**College**	**BENG**	
Module Leader	Rusal Ali Jabbar		**e-mail**	Rusulali915@gmail.com	
Module Leader's Acad. Title			**Module Leader's Qualification**		
Module Tutor			**e-mail**		
Peer Reviewer Name			**e-mail**		
Scientific Committee Approval Date			**Version Number**	1.0	

Relation with other Modules العلاقة مع المواد الدراسية الأخرى			
Prerequisite module	None	**Semester**	

Co-requsites module	None	**Semester**	

Module Aims, Learning Outcomes and Indicative Contents

أهداف المادة الدراسية ونتائج التعلم والمحتويات الإرشادية

Module Aims أهداف المادة الدراسية	1. In C++, a module is a self-contained unit of code that encapsulates related declarations and definitions. It aims to provide a way to organize and modularize code, making it easier to manage large-scale projects. The main aims of using modules in C++ are as follows: 2. Encapsulation: Modules help encapsulate related declarations and definitions, allowing for better organization and separation of concerns. This improves code readability and maintainability by making it easier to understand and navigate the codebase. 3. Dependency Management: Modules enable explicit control over dependencies between different parts of the code. By specifying which modules depend on each other, you can ensure that the necessary dependencies are resolved correctly during compilation. This helps avoid issues related to include file order and reduces the likelihood of name clashes. 4. Compilation Efficiency: Modules offer the potential for faster compilation times compared to traditional header file inclusion. Since modules are compiled separately and only need to be recompiled when their interface changes, incremental builds become more efficient. This can significantly reduce build times, especially for large projects. 5. Name Collision Avoidance: Modules introduce a new concept called module partitions, which help prevent name collisions. Each module has its own partition, and names defined within a module are local to that partition by default. This eliminates the need for complex header guards and reduces the chances of name clashes when different modules use the same identifiers. 6. Improved Tooling Support: Modules have been designed with better tooling support in mind. Compilers can provide enhanced features such as faster parsing, improved error diagnostics, and better code navigation and completion in integrated development environments (IDEs). This leads to a more efficient development experience for programmers. 7. Overall, the aim of using modules in C++ is to bring better organization, encapsulation, and dependency management to the language, resulting in more robust, maintainable, and efficient codebases.

Module Learning Outcomes مخرجات التعلم للمادة الدراسية	1. In C++, a module is a unit of encapsulation that contains a set of related declarations and definitions. The purpose of using modules in C++ is to improve the organization and structure of code, enhance compilation times, and provide better separation of interface and implementation. 2. Here are some common learning outcomes associated with understanding modules in C++: 3. Understand the concept of modules: Gain a clear understanding of what
	modules are and how they differ from traditional header files and source files. Learn about the benefits and motivations behind using modules in C++. 4. Define and import modules: Learn how to define your own modules by creating module interfaces and module implementations. Understand the process of importing modules into other parts of your codebase. 5. Use module interfaces: Explore the syntax and features of module interfaces. Understand how to define and export declarations (functions, classes, variables, etc.) from a module interface. 6. Use module implementations: Learn how to implement the functionality declared in a module interface. Understand how to define and import module implementations into your code. 7. Resolve dependencies: Understand how modules handle dependencies between different modules. Learn how to handle cyclic dependencies and circular references. 8. Improve build times: Explore how modules can significantly reduce compilation times by allowing for faster and more efficient incremental builds. Understand the impact of modules on build systems and integration with existing codebases. 9. Encapsulation and information hiding: Understand how modules promote encapsulation and information hiding by providing a clear separation between the interface and implementation. Learn how to design and structure modules to hide internal details. 10. Module organization and management: Gain knowledge about best practices for organizing and managing modules within a project. Understand how to structure your codebase to maximize reusability and maintainability. 11. Interoperability with existing code: Learn how to use modules alongside traditional header files and source files. Understand the interplay between modules and the pre-existing C++ codebase.

Indicative Contents	Indicative content includes the following. If you're looking to study C++, here are some indicative contents that you should consider covering: A 1 Introduction to C++: • History and features of C++ • Setting up a development environment 2 Basic Syntax: • Structure of a C++ program • Variables, data types, and operators • Input and output streams • Control flow statements (if-else, loops) 3 application B : It's important to note that this list is not exhaustive, and depending on your learning goals and requirements, you may need to explore additional topics. Additionally, practicing programming through hands-on coding exercises, projects, and solving algorithmic problems can greatly enhance your understanding of c++.

Learning and Teaching Strategies استراتيجيات التعلم والتعليم	
Strategies	Assessment is based on hand-in assignments, written exam, Case study, Quizzes, seminars, Practical testing , When it comes to learning and teaching C++, there are several strategies that can be effective. Here are some key strategies to consider: 1. Start with the basics: Begin by introducing the fundamental concepts of C++, such as variables, data types, control structures (loops and conditionals), functions, and object-oriented programming principles. It is important to establish a solid foundation before moving on to more advanced topics. 2. Hands-on coding: C++ is a practical programming language, so encourage students to engage in hands-on coding exercises and projects. Provide plenty of opportunities for students to write code, practice implementing algorithms, and solve programming problems. This helps reinforce their understanding of the language and builds their problem-solving skills. 3. Break down complex topics: C++ can be daunting for beginners due to its syntax and various features. Break down complex topics into smaller, more manageable parts. Provide clear explanations, examples, and step-by-step instructions to guide students through the learning process. 4. Interactive learning resources: Utilize interactive learning resources such as online coding platforms, C++ compilers, and integrated development environments (IDEs) with built-in tutorials and exercises. These resources can enhance the learning experience by providing immediate feedback and allowing students to experiment with code in a supportive environment. 5. Practice and repetition: Reinforce learning through practice and repetition. Assign coding exercises, small projects, and coding challenges to help students apply their knowledge and build confidence. Encourage them to solve problems independently while providing guidance and feedback along the way. 6. Real-world examples: Demonstrate real-world examples of how C++ is used in industry applications. This helps students see the practical relevance of what they are learning and motivates them to apply their skills beyond the classroom. 7. Peer collaboration: Encourage students to collaborate with their peers. Pair programming and group projects can foster a supportive learning environment where students can learn from each other, exchange ideas, and solve problems collectively. 8. Documentation and resources: Provide students with comprehensive documentation and additional learning resources, such as textbooks, online tutorials, and reference guides. These resources can serve as references for students to deepen their understanding and explore advanced topics at their own pace. 9. Code reviews and feedback: Review and provide constructive feedback on students' code. This helps identify areas for improvement, reinforces best practices, and encourages students to write clean, readable, and efficient code. 10. Stay updated: C++ is continuously evolving, with new features

	being introduced. Stay updated with the latest versions and trends in the language to ensure that your teaching materials and strategies align with current industry practices. Remember that every student has a unique learning style and pace. It's important to be flexible, patient, and adapt your teaching strategies to cater to different learning needs. Encourage a growth mindset, provide support, and foster a positive and inclusive learning environment to maximize student engagement and success.

Student Workload (SWL)
الحمل الدراسي للطالب

Structured SWL (h/sem) الحمل الدراسي المنتظم للطالب خلال الفصل	78	**Structured SWL (h/w)** الحمل الدراسي المنتظم للطالب أسبوعيا	5
Unstructured SWL (h/sem) الحمل الدراسي غير المنتظم للطالب خلال الفصل	22	**Unstructured SWL (h/w)** الحمل الدراسي غير المنتظم للطالب أسبوعيا	1.46
Total SWL (h/sem) الحمل الدراسي الكلي للطالب خلال الفصل	100		

Module Evaluation
تقييم المادة الدراسية

		Time/Number	Weight (Marks)	Week Due	Relevant Learning Outcome
Formative assessment	**Quizzes**	2	10% (10)	5, 10	LO #1, 2, 4 and 10
	Assignments	2	10% (10)	2, 12	LO # 3, 4, 8 and 9
	Projects / Lab.	1	10% (10)	Continuous	
	Report	1	10% (10)	13	LO # 5, 8 and 11
Summative assessment	**Midterm Exam**	3 hr	10% (10)	7	LO # 1-7
	Final Exam	3hr	50% (50)	16	All

Total assessment	100% (100 Marks)		

Delivery Plan (Weekly Syllabus)

المنهاج الاسبوعي النظري

	Material Covered
Week 1	Introduction - the programming
Week 2	Basics of Network Elements
Week 3	C++ review , control structures
Week 4	Slope intercept Equation Distance from point to the line
Week 5	Variables, loop statement
Week 6	Breaking statement , arrays one dimension
Week 7	Mid-term Exam 1
Week 8	Structures , enumerated data types
Week 9	Gis alug,c++
Week 10	arrays 2 dimension
Week 11	enumerated data types
Week 12	Response Function
Week 13	Domain And Range Functions
Week 14	Oop
Week 15	Mid-term Exam 2
Week 16	**Preparatory week before the final Exam**

Delivery Plan (Weekly Lab. Syllabus) المنهاج الاسبوعي للمختبر	
	Material Covered
Week 1	Lab 1: Introduction to c++
Week 2	Lab 2: Basics of Network wan , pan ,man .
Week 3	Lab 3: control structures
Week 4	Lab 4: Slope intercept Equation
Week 5	Lab 5: loop statement
Week 6	Lab 6 Structures pint
Week 7	Lab 7: arrays one dimension application

Learning and Teaching Resources مصادر التعلم والتدريس		
	Text	**Available in the Library?**
Required Texts		
Recommended Texts		

Grading Scheme مخطط الدرجات				
Group	**Grade**	التقدير	**Marks (%)**	**Definition**
Success Group (50 - 100)	**A -** Excellent	امتياز	90 - 100	Outstanding Performance
	B - Very Good	جدا جيد	80 - 89	Above average with some errors
	C - Good	جيد	70 - 79	Sound work with notable errors
	D - Satisfactory	متوسط	60 - 69	Fair but with major shortcomings
	E - Sufficient	مقبول	50 - 59	Work meets minimum criteria
Fail Group (0 – 49)	**FX –** Fail	راسب (قيد المعالجة)	(45-49)	More work required but credit awarded
	F – Fail	راسب	(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.