National University of Engineering (UNI)
School of Computer Science
Syllabus 2026-I

1. COURSE

CS112. Objects-oriented programming I (Mandatory)

2. GENERAL INFORMATION

2.1 Course	:	CS112. Objects-oriented programming I
2.2 Semester	:	2^{nd} Semester.
2.3 Credits	:	4
2.4 Hours	:	2 HT; 4 HP;
2.5 Duration of the period	:	16 weeks
2.6 Type of course	:	Mandatory
2.7 Learning modality	:	Face to face
2.8 Prerrequisites	:	CS111. Introduction to Computing. (1^{st} Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

This is the second course in the introductory Computer Science programming sequence. It incorporates a programming language transition as a pedagogical strategy and focuses on core Object-Oriented Programming (OOP) concepts in C++17/C++20, expanding on the foundations from the first course while introducing low-level programming concepts. The course provides a solid foundation for advanced topics covered in the subsequent course of the curriculum.

5. GOALS

- Introduce the student to the foundations of the object orientation paradigm in C++
- Develop skills in C++17/C++20 programming including templates and STL
- Understand memory management and low-level programming concepts
- Develop problem-solving skills using OOP principles
- Master intermediate C++ programming techniques

6. COMPETENCES

AG-C09) Design and Development of Solutions: Designs, implements, and evaluates solutions for complex computing problems. (Usage)

AG-C12) Uses systemic approaches to select, develop, apply, integrate, and manage secure computing technologies to achieve user objectives. (Usage)

7. TOPICS

Unit 1: Pragmática del Lenguaje (2 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Dominios de problemas y paradigma de programación • Criterios para un buen diseño de lenguaje de programación: – Principios de diseño de lenguajes como la ortogonalidad – Definir construcciones de control e iteración – Modularización de software grande • Brief review of programming paradigms • Comparison between functional and imperative programming • History of programming languages (emphasis on C and C++)	• Discuss the historical context of programming language paradigms [Familiarizarse]. • Evaluate language design trade-offs for specific applications [Evaluar].
Readings : [Str13], [Jos19]	

Unit 2: Traducción y Ejecución de Lenguajes (4 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Modelos de ejecución para JIT (Just-In-Time), compilador, intérprete • Uso de código intermedio, por ejemplo, bytecode • Ejecución como código nativo o dentro de una máquina virtual • Virtual machines and intermediate languages • Compilation vs. interpretation models	• Explicar y comprender las diferencias entre implementaciones de lenguaje compilado, JIT e interpretado, incluyendo los beneficios y limitaciones de cada una [Familiarizarse] • Diferenciar sintaxis y análisis (parsing) de semántica y evaluación [Usar] • Ilustrar ambigüedad en el análisis usando if-then-else anidado/expresión aritmética y mostrar resolución usando orden de precedencia [Evaluar]
Readings : [Str13], [DD17]	

Unit 3: Sistemas de Tipos I: Fundamentos (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Un tipo como un conjunto de valores junto con un conjunto de operaciones: – Tipos primitivos (por ejemplo, números, booleanos) – Tipos compuestos construidos a partir de otros tipos (por ejemplo, registros/estructuras, uniones, arreglos, listas, funciones, referencias usando operaciones de conjuntos) • Tipos genéricos (polimorfismo paramétrico): – Definición y ventajas del polimorfismo: paramétrico, de subtipado (subtyping), de sobrecarga (overloading) y de coerción – Comparación de tipos monomórficos y polimórficos – Comparación con polimorfismo ad-hoc (sobrecarga) y polimorfismo de subtipos – Parámetros genéricos y tipado – Uso de bibliotecas genéricas como colecciones – Comparación con polimorfismo ad hoc (sobrecarga) y polimorfismo de subtipos – Polimorfismo prescriptivo vs descriptivo – Modelos de implementación de tipos polimórficos – Subtipado • Declaration models (linking, visibility, scope, lifetime) • Overview of type checking in C++ • Type conversion and casting in C++	• Definir y usar piezas de programa (como funciones, clases, métodos) que usen tipos genéricos, incluyendo para colecciones [Usar] • Discutir las diferencias entre genéricos, subtipado y sobrecarga [Familiarizarse] • Explicar múltiples beneficios y limitaciones del tipado estático al escribir, mantener y depurar software [Evaluar]
Readings : [Str13], [Jos19]	

Unit 4: Programación Orientada a Objetos I: Fundamentos (14 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Definición de clases: campos, métodos y constructores. • Subclases, herencia (incluyendo herencia múltiple) y sobrescritura de métodos. • Despacho dinámico: definición de llamada a método. • Subtipado: – Polimorfismo de subtipos; conversiones ascendentes (*upcasts*) implícitas en lenguajes tipados. – Noción de reemplazo conductual: los subtipos actúan como el supertipo. – Relación entre subtipado y herencia. • Classes and objects in C++ • Constructors and destructors • Access specifiers and encapsulation • Inheritance hierarchies • Polymorphism and virtual functions	• Componer una clase a través del diseño, implementación y prueba para cumplir con los requisitos de comportamiento [Usar] • Construir una jerarquía de clases simple utilizando subclases que permita reutilizar código para subclases distintas [Usar] • Predecir y validar el flujo de control en un programa usando despacho dinámico [Evaluar] • Design C++ classes with proper encapsulation [Usar]. • Implement inheritance hierarchies for code reuse [Usar].
Readings : [Str13], [Jos19]	

Unit 5: Modelo de Ejecución y Memoria de Sistemas (12 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Acceso directo, indirecto e indexado a ubicación de memoria • Representación en tiempo de ejecución de abstracciones de datos como variables, arreglos, vectores, registros, elementos de datos basados en punteros como listas enlazadas y árboles, y objetos • Asignación y acceso de bajo nivel a estructuras de datos de alto nivel como tipos de datos básicos, arreglos n-dimensionales, vectores, registros y objetos • Retorno de procedimiento como mecanismo de desasignación automática para elementos de datos locales en la pila • Gestión manual de memoria: asignar, desasignar y reutilizar memoria del montón • Gestión automática de memoria: recolección de basura (garbage collection) como técnica automatizada usando la noción de alcanzabilidad (reachability) • Memory layout (stack, heap, static) • Pointer arithmetic and operations • Dynamic memory allocation (new/delete) • Memory leaks and dangling pointers • Smart pointers (unique_ptr, shared_ptr)	• Explicar por qué ocurren las fugas de memoria (memory leaks) y los problemas de punteros colgantes (dangling pointers), y qué puede hacer un programador para evitarlos/corregirlos [Usar] • Explicar cómo las implementaciones de lenguajes de programación típicamente organizan la memoria en secciones de datos globales, texto, montón y pila, y cómo características como recursión y gestión de memoria se mapean a este modelo de memoria [Evaluar] • Explicar cómo se ejecuta una construcción central del lenguaje, como abstracciones de datos y abstracciones de control [Evaluar] • Implement RAII for resource management [Usar]. • Debug memory-related errors in C++ programs [Usar].
Readings : [Str13], [DD17]	

Unit 6: Programación Funcional (10 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Expresiones lambda y evaluación: – Enlace (binding) de variables y reglas de ámbito (scope). – Paso de parámetros. – Expresiones lambda anidadas y orden de reducción. • Usar funciones de orden superior (tomar, devolver y almacenar funciones). • Function templates and class templates • Template specialization • Lambda expressions in C++11/14 • Standard Template Library concepts	• Desarrollar funciones útiles que tomen y devuelvan otras funciones [Usar] • Interpretar correctamente las variables y el ámbito léxico en un programa que usa clausuras de funciones [Evaluar] • Usar mecanismos de encapsulación funcional como clausuras e interfaces modulares [Usar] • Implement generic functions using templates [Usar]. • Use lambda expressions for functional programming [Usar].
Readings : [Str13], [Jos19]	

Unit 7: Programación Orientada a Objetos II: Encapsulación, Subtipado y Reflexión (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Clases de colecciones, iteradores y otros componentes comunes de biblioteca. • Sequence containers (vector, list, deque) • Associative containers (set, map, unordered_set) • Container adapters (stack, queue, priority_queue) • Iterators and algorithms • Functors and predicates	• Usar clases de colecciones e iteradores efectivamente para resolver un problema [Usar] • Definir y usar iteradores y otras operaciones en agregados, incluyendo operaciones que toman funciones como argumentos, en múltiples lenguajes de programación, seleccionando los idiomas más naturales para cada lenguaje [Usar] • Comparar y contrastar los mecanismos para definir y proteger elementos de datos dentro de los enfoques procedural, funcional y orientado a objetos [Evaluar] • Select appropriate STL containers for specific needs [Usar].
Readings : [Str13], [Jos19]	

Unit 8: Construcciones de Programación Avanzadas (6 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Abstracciones orientadas a objetos: herencia múltiple, mixins, rasgos (traits), multimétodos (multimethods) • Metaprogramación: macros, programación generativa, desarrollo basado en modelos (model-based development) • Operator overloading • Move semantics and rvalue references • Exception handling in C++ • Namespaces and modular programming	• Usar varias construcciones e idiomas de programación avanzados correctamente [Usar] • Discutir cómo varias construcciones de programación avanzadas tienen como objetivo mejorar la estructura del programa, la calidad del software y la productividad del programador [Evaluar] • Discutir cómo varias construcciones de programación avanzadas interactúan con la definición e implementación de otras características del lenguaje [Familiarizarse] • Implement move semantics for performance optimization [Usar].
Readings : [Str13], [Jos19]	

Unit 9: Scripting de Shell (4 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Comandos del sistema: – Interfaz con sistemas operativos • Tuberías (piping) • Abstracción de archivo y operadores • Programas y procesos • Automating compilation and testing of C++ programs through scripts (Makefiles, build/test scripts)	• Crear y ejecutar scripts automatizados para gestionar varias tareas del sistema [Usar]
Readings : [RB05]	

8. WORKPLAN

8.1 Methodology

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

8.2 Theory Sessions

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

8.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

9. EVALUATION SYSTEM

********* EVALUATION MISSING ********

10. BASIC BIBLIOGRAPHY

[RB05] Arnold Robbins and Nelson H.F. Beebe. *Classic Shell Scripting*. O'Reilly Media, 2005.

[Str13] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, 2013.

[DD17] Harvey Deitel and Paul Deitel. *C++17: The Complete Guide*. Pearson, 2017.

[Jos19] Nicolai M. Josuttis. *C++17: The Complete Guide*. Pearson Education, 2019.