UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE CIENCIAS MATEMÁTICAS

SÍLABO

1. INFORMACIÓN GENERAL

1.1 Curso	:	Programación Orientada a Objetos I
1.2 Código	:	CS112
1.3 Programa Académico	:	Computación Científica
1.4 Créditos	:	4
1.5 Prerrequisitos	:	CS111. Introducción a la Programación. (1^{er} Sem)
1.6 Tipo de curso	:	Obligatorio
1.7 Área de Estudios	:	Estudios Generales
1.8 Número de Semanas	:	16
1.9 Horas por semana	:	2 HT; 4 HP;
1.10 Ciclo	:	I
1.11 Modalidad de aprendizaje	:	Presencial
1.12 Periodo Académico	:	2026-I
1.13 Inicio del semestre	:	
1.14 Fin del semestre	:	

2. PROFESORES

Atención previa coordinación con el profesor

3. INTRODUCCIÓN AL CURSO

Este es el segundo curso en la secuencia introductoria de programación en Ciencia de la Computación. Incorpora una transición de lenguaje de programación como estrategia pedagógica y se enfoca en los conceptos centrales de Programación Orientada a Objetos (POO) en C++17/C++20, expandiendo las bases del primer curso mientras introduce conceptos de programación de bajo nivel. El curso proporciona una base sólida para los temas avanzados cubiertos en el curso subsiguiente del plan de estudios.

4. OBJETIVOS

- Introducir al estudiante a los fundamentos del paradigma de orientación a objetos en C++.
- Desarrollar habilidades en programación C++17/C++20, incluyendo plantillas y STL.
- Comprender la gestión de memoria y conceptos de programación de bajo nivel.
- Desarrollar habilidades de resolución de problemas usando principios de POO.
- Dominar técnicas de programación C++ intermedias.

5. RESULTADOS DEL ESTUDIANTE

ABET-2) Habilidad para formular o diseñar un sistema, proceso, procedimiento o programa para el propósito previsto. (Usage)

ABET-3) Habilidad para desarrollar y llevar a cabo experimentos o probar hipótesis, analizar e interpretar datos y utilizar el juicio científico para obtener conclusiones. (Usage)

6. TEMAS

Unidad 1: Pragmática del Lenguaje (2 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*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 ▪ Breve revisión de paradigmas de programación. ▪ Comparación entre programación funcional e imperativa. ▪ Historia de los lenguajes de programación (énfasis en C y C++).	▪ Discutir el contexto histórico de los paradigmas de lenguajes de programación [Familiarizarse]. ▪ Evaluar las compensaciones en el diseño de lenguajes para aplicaciones específicas [Evaluar].
Lecturas : [**Stroustrup2013**], [**Josuttis2019**]	

Unidad 2: Traducción y Ejecución de Lenguajes (4 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*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 ▪ Máquinas virtuales y lenguajes intermedios. ▪ Modelos de compilación vs. interpretación.	▪ 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]
Lecturas : [**Stroustrup2013**], [**Deitel2017**]	

Unidad 3: Fundamentos de Sistemas de Tipos (8 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*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 ▪ Modelos de declaración (enlace, visibilidad, alcance, tiempo de vida). ▪ Visión general de la verificación de tipos en C++. ▪ Conversión de tipos y casting en 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]
Lecturas : [**Stroustrup2013**], [**Josuttis2019**]	

Unidad 4: Fundamentos de Programación Orientada a Objetos (14 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*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. ▪ Clases y objetos en C++. ▪ Constructores y destructores. ▪ Especificadores de acceso y encapsulamiento. ▪ Jerarquías de herencia. ▪ Polimorfismo y funciones virtuales.	▪ 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] ▪ Diseñar clases en C++ con encapsulamiento adecuado [Usar]. ▪ Implementar jerarquías de herencia para reutilización de código [Usar].
Lecturas : [**Stroustrup2013**], [**Josuttis2019**]	

Unidad 5: Modelo de Ejecución y Memoria de Sistemas (12 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*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) ▪ Distribución de memoria (pila, montón, estática). ▪ Aritmética de punteros y operaciones. ▪ Asignación dinámica de memoria (new/delete). ▪ Fugas de memoria y punteros colgantes. ▪ Punteros inteligentes (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] ▪ Implementar RAII para la gestión de recursos [Usar]. ▪ Depurar errores relacionados con memoria en programas C++ [Usar].
Lecturas : [**Stroustrup2013**], [**Deitel2017**]	

Unidad 6: Programación Funcional (10 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*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).
▪ Plantillas de funciones y plantillas de clases.
▪ Especialización de plantillas.
▪ Expresiones lambda en C++11/14.
▪ Conceptos de la Biblioteca de Plantillas Estándar (STL).
	▪ 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]
	▪ Implementar funciones genéricas usando plantillas [Usar].
	▪ Usar expresiones lambda para programación funcional [Usar].

Lecturas : [**Stroustrup2013**], [**Josuttis2019**]	

Unidad 7: Programación Orientada a Objetos: Encapsulación, Subtipado y Reflexión (8 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ Clases de colecciones, iteradores y otros componentes comunes de biblioteca. ▪ Contenedores de secuencia (vector, list, deque). ▪ Contenedores asociativos (set, map, unordered_set). ▪ Adaptadores de contenedor (stack, queue, priority_queue). ▪ Iteradores y algoritmos. ▪ Funtores y predicados.	▪ 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] ▪ Seleccionar contenedores STL apropiados para necesidades específicas [Usar].
Lecturas : [**Stroustrup2013**], [**Josuttis2019**]	

Unidad 8: Construcciones de Programación Avanzadas (6 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ Abstracciones orientadas a objetos: herencia múltiple, mixins, rasgos (traits), multimétodos (multimethods) ▪ Metaprogramación: macros, programación generativa, desarrollo basado en modelos (model-based development) ▪ Sobrecarga de operadores. ▪ Semántica de movimiento y referencias a valor-r. ▪ Manejo de excepciones en C++. ▪ Espacios de nombres y programación modular.	▪ 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] ▪ Implementar semántica de movimiento para optimización de rendimiento [Usar].
Lecturas : [**Stroustrup2013**], [**Josuttis2019**]	

Unidad 9: Scripting de Shell (4 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ Comandos del sistema: • Interfaz con sistemas operativos ▪ Tuberías (piping) ▪ Abstracción de archivo y operadores ▪ Programas y procesos ▪ Automatización de la compilación y pruebas de programas C++ mediante scripts (Makefiles, scripts de build/test).	▪ Crear y ejecutar scripts automatizados para gestionar varias tareas del sistema [Usar]
Lecturas : [**Robbins2005**]	

7. PLAN DE TRABAJO

7.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

7.2 Sesiones Teóricas

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

7.3 Sesiones Prácticas

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

8. SISTEMA DE EVALUACIÓN

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

9. BIBLIOGRAFÍA BÁSICA