UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE CIENCIAS MATEMÁTICAS

SÍLABO

1. INFORMACIÓN GENERAL

1.1 Curso	:	Programación Orientada a Objetos II
1.2 Código	:	CS113
1.3 Programa Académico	:	Computación Científica
1.4 Créditos	:	3
1.5 Prerrequisitos	:	CS112. Programación Orientada a Objetos I. (2^{do} 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	:	1 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 tercer curso en la secuencia de cursos introductorios en ciencia de la computación. Este curso explora características avanzadas de C++17/C++20 centrándose en el desarrollo de sistemas de alto rendimiento. Los temas clave incluyen metaprogramación con plantillas, semántica de movimiento, reenvío perfecto, optimización RAII, patrones de herencia múltiple/virtual, concurrencia con std::thread y async/await, programación sin bloqueo, paradigmas modernos de C++ como CRTP y plantillas de expresión, e interfaz de C++ con otros lenguajes. El curso prepara a los estudiantes para motores de juego, computación de alto rendimiento y desarrollo embebido donde C++ es dominante.

4. OBJETIVOS

- Dominar características avanzadas de C++17/C++20, incluyendo plantillas y metaprogramación.
- Desarrollar habilidades en programación concurrente y de sistemas.
- Comprender técnicas de optimización de rendimiento.
- Prepararse para aplicaciones de computación de alto rendimiento.
- Dominar la metaprogramación con plantillas y el cómputo en tiempo de compilación.

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: 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 asociativos (std::set, std::map, variantes no ordenadas). ▪ Adaptadores de contenedor (stack, queue, priority_queue). ▪ Algoritmos avanzados de STL. ▪ Funtores y predicados. ▪ Categorías de iteradores y rasgos.	▪ 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 [Evaluar] ▪ Comparar y contrastar los beneficios y costos/impacto de usar herencia (subclases) y composición (específicamente, cómo basar la composición en funciones de orden superior) [Usar] ▪ Implementar asignadores personalizados para contenedores STL [Usar].
Lecturas : [**Stroustrup2013**], [**Josuttis2019**], [**Vandevoorde2017**]	

Unidad 2: Sistemas de Tipos: Polimorfismo y Tipado Complementario (12 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ 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 ▪ SFINAE (Fallo de Sustitución No es un Error): fundamento histórico de la programación genérica restringida en C++, base de bibliotecas anteriores a C++20. ▪ Concepts (C++20): sintaxis declarativa para restringir parámetros de plantilla, sucesor moderno de SFINAE con mejores diagnósticos de error. ▪ if constexpr (C++17): ramificación en tiempo de compilación, reemplazo de gran parte del despacho por etiquetas (tag dispatch) antes resuelto con SFINAE. ▪ Reenvío perfecto y referencias universales. ▪ Plantillas variádicas. ▪ Cómputo en tiempo de compilación con constexpr y consteval.	▪ Definir y usar piezas de programa (como funciones, clases, métodos) que usen tipos genéricos, incluyendo para colecciones [Evaluar] ▪ Discutir las diferencias entre genéricos, subtipado y sobrecarga [Evaluar] ▪ Aplicar Concepts para restringir parámetros de plantilla y mejorar diagnósticos de error [Usar]. ▪ Distinguir cuándo usar Concepts, if constexpr o SFINAE heredado al mantener código C++ existente [Evaluar]. ▪ Implementar reenvío perfecto para paso eficiente de argumentos [Usar].
Lecturas : [**Stroustrup2013**], [**Vandevoorde2017**], [**StroustrupTour2022**]	

Unidad 3: Gestión de Memoria (8 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ 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) ▪ Lvalues y rvalues en C++11/14/17/20. ▪ Constructores de movimiento y operadores de asignación de movimiento. ▪ Referencias a valor-r y colapso de referencias. ▪ Patrón RAII (Adquisición de Recursos es Inicialización). ▪ Personalización de punteros inteligentes. ▪ Detección y prevención de fugas de memoria.	▪ 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 semántica de movimiento para optimización de recursos [Usar]. ▪ Diseñar clases RAII para gestión de recursos segura ante excepciones [Usar].
Lecturas : [**Meyers2014**], [**Stroustrup2013**]	

Unidad 4: Computación Paralela y Distribuida (12 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**

▪ Seguridad (safety) y vivacidad (liveness):
  • Condiciones de carrera (race conditions)
  • Dependencias/precondiciones
  • Modelos de fallos (fault models)
  • Terminación
▪ Modelos de programación:
  • Modelo de actores (*actor models*)
  • Modelos procedurales y reactivos
  • Modelos de programación síncrona/asíncrona
  • Paralelismo de datos (*data parallelism*)
▪ Comunicación y coordinación:
  • Mutexes
  • Paso de mensajes (*message-passing*)
  • Memoria compartida (*shared memory*)
  • Cobegin-coend
  • Monitores
  • Canales (*channels*)
  • Hilos (*threads*)
  • Guardas (*guards*)
▪ std::thread y gestión de hilos.
▪ std::jthread (C++20): unión automática (RAII) y cancelación cooperativa vía stop_token.
▪ Mutexes, candados y variables de condición.
▪ Mecanismos de sincronización de C++20: std::latch, std::barrier, std::counting_semaphore.
▪ Operaciones atómicas y programación sin bloqueo.
▪ std::async y futures.
▪ std::execution y el modelo de senders-receivers (C++26, experimental): panorama hacia dónde evoluciona la concurrencia estándar en C++, presentado a nivel conceptual, sin evaluación en laboratorio dada su disponibilidad limitada (parcial en GCC).
▪ Patrones de diseño seguros para hilos.
▪ Carreras de datos y prevención de interbloqueos.
	▪ Implementar programas concurrentes correctos usando múltiples modelos de programación, como memoria compartida, actores, futuros, construcciones de sincronización y primitivas de paralelismo de datos [Usar]
	▪ Usar construcciones de sincronización como métodos monitorizados/sincronizados en un programa simple [Usar]
	▪ Usar un modelo de paso de mensajes para analizar un protocolo de comunicación [Evaluar]
	▪ Implementar estructuras de datos sin bloqueo usando operaciones atómicas [Usar].
	▪ Diferenciar std::thread de std::jthread y justificar cuándo usar cada uno [Usar].
	▪ Analizar la seguridad en hilos de algoritmos concurrentes [Evaluar].

Lecturas : [**Williams2019**], [**Stroustrup2013**]	

Unidad 5: Construcciones de Programación Avanzadas (8 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) ▪ Herencia múltiple y herencia virtual. ▪ Patrón CRTP (Patrón de Plantilla Curiosamente Recursivo). ▪ deducing this (C++23): simplifica CRTP al eliminar la necesidad de static_cast<Derived&>(*this) para polimorfismo estático. ▪ Plantillas de expresión. ▪ Rasgos de tipos e introspección en tiempo de compilación. ▪ Diseño basado en políticas. ▪ Diseño de interfaz para bibliotecas.	▪ 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 [Evaluar] ▪ Implementar CRTP para polimorfismo estático [Usar]. ▪ Diseñar interfaces de biblioteca extensibles usando diseño basado en políticas [Usar].
Lecturas : [**Stroustrup2013**], [**Gregor2017**]	

Unidad 6: Modelo de Ejecución y Memoria de Sistemas (8 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ 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 ▪ Estructuras de datos para traducción, ejecución, traducción y movilidad de código como pila, montón (heap), alias (compartición usando punteros), secuencia indexada y cadena ▪ Implementación del despacho dinámico introducido en POO I: tabla de funciones virtuales (vtable), puntero vtable (vptr), y su costo en memoria y rendimiento frente al despacho estático. ▪ Interfaz de C++ con otros lenguajes (FFI). ▪ Llamadas al sistema e interacción con el SO. ▪ E/S mapeada en memoria. ▪ Perfilado de rendimiento y evaluación comparativa. ▪ Técnicas de optimización para HPC.	▪ Explicar cómo se ejecuta una construcción central del lenguaje, como abstracciones de datos y abstracciones de control [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 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] ▪ Perfilar y optimizar programas C++ para rendimiento [Usar]. ▪ Explicar cómo el compilador implementa el polimorfismo en tiempo de ejecución mediante vtables [Evaluar]. ▪ Interfazar código C++ con bibliotecas C [Usar].
Lecturas : [**Stroustrup2013**], [**Williams2019**]	

Unidad 7: 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) ▪ Patrones creacionales, estructurales y de comportamiento. ▪ Patrones singleton seguros para hilos. ▪ Modismos de diseño modernos en C++. ▪ Inyección de dependencias e inversión de control.	▪ 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 patrones de diseño seguros para hilos [Usar]. ▪ Evaluar compensaciones arquitectónicas en sistemas C++ [Evaluar].
Lecturas : [**Gamma1994**], [**Stroustrup2013**]	

Unidad 8: Construcciones de Programación Avanzadas (6 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ Abstracciones de control: manejo de excepciones, continuaciones (continuations), mónadas (monads). ▪ Optimizaciones del compilador y banderas. ▪ Metodologías de evaluación comparativa. ▪ Estructuras de datos amigables con la caché. ▪ Predicción de bifurcación y alineación. ▪ Reducción del tiempo de compilación. ▪ Integración continua para pruebas de rendimiento.	▪ 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] ▪ Aplicar técnicas de optimización para mejorar el rendimiento del programa [Usar]. ▪ Interpretar resultados de evaluación comparativa para guiar decisiones de optimización [Usar].
Lecturas : [**Meyers2014**], [**Stroustrup2013**]	

Unidad 9: Control de Versiones y CI/CD (4 horas)	
Resultados esperados: ABET-2,ABET-3	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**

▪ Automatización de procesos de software:
  • Sistemas de construcción (*build systems*): el valor de builds rápidos, herméticos y reproducibles, comparar/contrastar enfoques para construir un proyecto.
  • Integración Continua (CI): el uso de automatización y pruebas automatizadas para hacer una validación preliminar de que la revisión actual del tronco se construye y pasa las pruebas (básicas).
  • Despliegue Continuo (CD): el uso de automatización para liberar de forma automática cada cambio que supera las pruebas hacia el entorno de producción, garantizando entregas frecuentes y confiables.
  • Gestión de dependencias: actualización de dependencias externas/aguas arriba, gestión de paquetes, SemVer.
▪ Sistemas de construcción para C++: CMake, el estándar de facto de la industria.
▪ Alternativas modernas: xmake, Meson y Bazel – comparación de enfoques declarativos frente a basados en Lua/Python.
▪ Organización de proyectos multi-archivo: unidades de traducción, cabeceras, bibliotecas estáticas y dinámicas.
▪ Gestión de dependencias en C++ (vcpkg, Conan).
	▪ Comprender el uso de sistemas de CI/CD como una fuente de verdad para el estado del código compartido del equipo (éxito en la construcción y pruebas) [Familiarizarse]
	▪ Configurar un proyecto C++ multi-archivo usando un sistema de construcción declarativo [Usar].
	▪ Comparar CMake con al menos una alternativa moderna (xmake, Meson o Bazel) en velocidad de compilación y reproducibilidad [Evaluar].

Lecturas : [**Stroustrup2013**], [**StroustrupTour2022**]	

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