National University of Engineering (UNI)
School of Artificial Intelligence
Syllabus 2027-I

1. COURSE

CS113. Objects-oriented programming II (Mandatory)

2. GENERAL INFORMATION

2.1 Course	:	CS113. Objects-oriented programming II
2.2 Semester	:	3rd 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	:	CS112. Objects-oriented programming I. (2nd Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

This is the third course in the sequence of introductory courses in computer science. This course explores advanced C++17/C++20 features focusing on high-performance systems development. Key topics include template metaprogramming, move semantics, perfect forwarding, RAII optimization, multiple/virtual inheritance patterns, concurrency with std::thread and async/await, lock-free programming, modern C++ paradigms such as CRTP and expression templates, and interfacing C++ with other languages. The course prepares students for game engines, high-performance computing, and embedded development where C++ dominates.

5. GOALS

- Master advanced C++17/C++20 features including templates and metaprogramming
- Develop skills in concurrent and systems programming
- Understand performance optimization techniques
- Prepare for high-performance computing applications
- Master template metaprogramming and compile-time computation

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. (Assessment)

7. TOPICS

Unit 1: Object-Oriented Programming: Encapsulation, Subtyping, and Reflection (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Collection classes, iterators, and other common library components. • Associative containers (std::set, std::map, unordered variants) • Container adapters (stack, queue, priority_queue) • Advanced STL algorithms • Functors and predicates • Iterator categories and traits	• Use collection classes and iterators effectively to solve a problem [Usar] • Define and use iterators and other operations on aggregates, including operations that take functions as arguments, in multiple programming languages, selecting the most natural idioms for each language [Evaluar] • Compare and contrast the benefits and costs/impact of using inheritance (subclasses) and composition (specifically, how to base composition on higher order functions) [Usar] • Implement custom allocators for STL containers [Usar].
Readings : [**Stroustrup2013**], [**Josuttis2019**], [**Vandevoorde2017**]	

Unit 2: Type Systems: Polymorphism and Complementary Typing (12 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Generic types (parametric polymorphism): – Definition and advantages of polymorphism: parametric, subtyping, overloading, and coercion – Comparison of monomorphic and polymorphic types – Comparison with ad-hoc polymorphism (overloading) and subtype polymorphism – Generic parameters and typing – Use of generic libraries such as collections – Comparison with ad hoc polymorphism (overloading) and subtype polymorphism – Prescriptive vs descriptive polymorphism – Implementation models of polymorphic types – Subtyping • SFINAE (Substitution Failure Is Not An Error): historical foundation of constrained generic programming in C++, the basis of pre-C++20 libraries. • Concepts (C++20): declarative syntax to constrain template parameters, the modern successor to SFINAE with better error diagnostics. • if constexpr (C++17): compile-time branching, replacing much of the tag dispatch previously handled with SFINAE. • Perfect forwarding and universal references • Variadic templates • Compile-time computation with constexpr and consteval	• Define and use program pieces (such as functions, classes, methods) that use generic types, including for collections [Evaluar] • Discuss the differences among generics, subtyping, and overloading [Evaluar] • Apply Concepts to constrain template parameters and improve error diagnostics [Usar]. • Distinguish when to use Concepts, if constexpr, or legacy SFINAE when maintaining existing C++ code [Evaluar]. • Implement perfect forwarding for efficient argument passing [Usar].
Readings : [**Stroustrup2013**], [**Vandevoorde2017**], [**StroustrupTour2022**]	

Unit 3: Memory Management (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Low level allocation and accessing of high-level data structures such as basic data types, n-dimensional array, vector, record, and objects • Return from procedure as automatic deallocation mechanism for local data elements in the stack • Manual memory management: allocating, deallocating, and reusing heap memory • Automated memory management: garbage collection as an automated technique using the notion of reachability • Lvalues and rvalues in C++11/14/17/20 • Move constructors and move assignment operators • Rvalue references and reference collapsing • Resource Acquisition Is Initialization (RAII) pattern • Smart pointer customization • Memory leak detection and prevention	• Explain why memory leaks and dangling pointer problems occur, and what can be done by a programmer to avoid/fix them [Usar] • Explain how programming language implementations typically organize memory into global data, text, heap, and stack sections and how features such as recursion and memory management map to this memory model [Evaluar] • Explain how a core language construct, such as data abstractions and control abstractions, is executed [Evaluar] • Implement move semantics for resource optimization [Usar]. • Design RAII classes for exception-safe resource management [Usar].
Readings : [**Meyers2014**], [**Stroustrup2013**]	

Unit 4: Parallel and Distributed Computing (12 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Safety and liveness: – Race conditions – Dependencies/preconditions – Fault models – Termination • Programming models: – Actor models – Procedural and reactive models – Synchronous/asynchronous programming models – Data parallelism • Communication and coordination: – Mutexes – Message-passing – Shared memory – Cobegin-coend – Monitors – Channels – Threads – Guards • std::thread and thread management • std::jthread (C++20): automatic joining (RAII) and cooperative cancellation via stop_token • Mutexes, locks, and condition variables • C++20 synchronization primitives: std::latch, std::barrier, std::counting_semaphore • Atomic operations and lock-free programming • std::async and futures • std::execution and the senders-receivers model (C++26, experimental): a preview of where standard C++ concurrency is heading, presented at a conceptual level only, not evaluated in lab given its limited availability (partial in GCC) • Thread-safe design patterns • Data races and deadlock prevention	• Implement correct concurrent programs using multiple programming models, such as shared memory, actors, futures, synchronization constructs, and data-parallelism primitives [Usar] • Use synchronization constructions such as monitor/synchronized methods in a simple program [Usar] • Use a message-passing model to analyze a communication protocol [Evaluar] • Implement lock-free data structures using atomics [Usar]. • Differentiate std::thread from std::jthread and justify when to use each [Usar]. • Analyze thread-safety of concurrent algorithms [Evaluar].
Readings : [**Williams2019**], [**Stroustrup2013**]	

Unit 5: Advanced Programming Constructs (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Object-oriented abstractions: multiple inheritance, mixins, traits, multimethods • Metaprogramming: macros, generative programming, model-based development • Multiple inheritance and virtual inheritance • CRTP (Curiously Recurring Template Pattern) • deducing this (C++23): simplifies CRTP by removing the need for static_cast<Derived&>(*this) for static polymorphism • Expression templates • Type traits and compile-time introspection • Policy-based design • Interface design for libraries	• Use various advanced programming constructs and idioms correctly [Usar] • Discuss how various advanced programming constructs aim to improve program structure, software quality, and programmer productivity [Evaluar] • Discuss how various advanced programming constructs interact with the definition and implementation of other language features [Evaluar] • Implement CRTP for static polymorphism [Usar]. • Design extensible library interfaces using policy-based design [Usar].
Readings : [**Stroustrup2013**], [**Gregor2017**]	

Unit 6: Systems Execution and Memory Model (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Run-time representation of data abstractions such as variables, arrays, vectors, records, pointer-based data elements such as linked-lists and trees, and objects • Data structures for translation, execution, translation, and code mobility such as stack, heap, aliasing (sharing using pointers), indexed sequence and string • Implementation of the dynamic dispatch introduced in OOP I: virtual function table (vtable), vtable pointer (vptr), and its memory/performance cost versus static dispatch • Interfacing C++ with other languages (FFI) • System calls and OS interaction • Memory-mapped I/O • Performance profiling and benchmarking • Optimization techniques for HPC	• Explain how a core language construct, such as data abstractions and control abstractions, is executed [Usar] • Explain how programming language implementations typically organize memory into global data, text, heap, and stack sections and how features such as recursion and memory management map to this memory model [Evaluar] • Explain why memory leaks and dangling pointer problems occur, and what can be done by a programmer to avoid/fix them [Usar] • Profile and optimize C++ programs for performance [Usar]. • Explain how the compiler implements runtime polymorphism via vtables [Evaluar]. • Interface C++ code with C libraries [Usar].
Readings : [**Stroustrup2013**], [**Williams2019**]	

Unit 7: Advanced Programming Constructs (6 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Object-oriented abstractions: multiple inheritance, mixins, traits, multimethods • Creational, structural, and behavioral patterns • Thread-safe singleton patterns • Modern C++ design idioms • Dependency injection and inversion of control	• Use various advanced programming constructs and idioms correctly [Usar] • Discuss how various advanced programming constructs aim to improve program structure, software quality, and programmer productivity [Evaluar] • Discuss how various advanced programming constructs interact with the definition and implementation of other language features [Familiarizarse] • Implement thread-safe design patterns [Usar]. • Evaluate architectural trade-offs in C++ systems [Evaluar].
Readings : [**Gamma1994**], [**Stroustrup2013**]	

Unit 8: Advanced Programming Constructs (6 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Control abstractions: exception handling, continuations, monads. • Compiler optimizations and flags • Benchmarking methodologies • Cache-friendly data structures • Branch prediction and alignment • Reducing compilation time • Continuous integration for performance testing	• Use various advanced programming constructs and idioms correctly [Usar] • Discuss how various advanced programming constructs aim to improve program structure, software quality, and programmer productivity [Evaluar] • Discuss how various advanced programming constructs interact with the definition and implementation of other language features [Familiarizarse] • Apply optimization techniques to improve program performance [Usar]. • Interpret benchmark results to guide optimization decisions [Usar].
Readings : [**Meyers2014**], [**Stroustrup2013**]	

Unit 9: Version Control and CI/CD (4 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	**Learning Outcomes**
• Software process automation: – Build systems - the value of fast, hermetic, reproducible builds, compare/contrast approaches to building a project. – Continuous Integration (CI) - the use of automation and automated tests to do preliminary validation that the current head/trunk revision builds and passes (basic) tests. – Continuous Deployment (CD) - the use of automation to automatically release every change that passes the automated tests to the production environment, ensuring frequent and reliable deliveries. – Dependency management - updating external/upstream dependencies, package management, SemVer. • Build systems for C++: CMake, the de facto industry standard • Modern alternatives: xmake, Meson, and Bazel – comparing declarative approaches versus Lua/Python-based ones • Organizing multi-file projects: translation units, headers, static and dynamic libraries • Dependency management in C++ (vcpkg, Conan)	• Understand the use of CI/CD systems as a ground-truth for the state of the team's shared code (build and test success) [Familiarizarse] • Configure a multi-file C++ project using a declarative build system [Usar]. • Compare CMake with at least one modern alternative (xmake, Meson, or Bazel) in terms of build speed and reproducibility [Evaluar].
Readings : [**Stroustrup2013**], [**StroustrupTour2022**]	

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