NATIONAL UNIVERSITY OF SAN MARCOS

FACULTY OF MATHEMATICAL SCIENCES

SYLLABUS

1. GENERAL INFORMATION

1.1 Course name	:	Objects-oriented programming I
1.2 Code	:	CS112
1.3 Academic Program	:	Scientific Computing
1.4 Credits	:	4
1.5 Prerrequisites	:	CS111. Introduction to Programming. (1st Sem)
1.6 Type of Course	:	Mandatory
1.7 Area of Studies	:	Estudios Generales
1.8 Number of Weeks	:	16
1.9 Hours per week	:	2 HT; 4 HP;
1.10 Cycle	:	I
1.11 Learning modality	:	Face to face
1.12 Academic Period	:	2026-I
1.13 Semester Start Date	:	
1.14 Semester End Date	:	

2. PROFESSORS

Meetings after coordination with the professor

3. 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.

4. 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

5. COMPETENCES

ABET-2) An ability to formulate or design a system, process, procedure, or program for the intended purpose. (Usage)

ABET-3) An ability to develop and conduct experiments or test hypotheses, analyze and interpret data, and use scientific judgment to draw conclusions. (Usage)

6. TOPICS

Unit 1: Language Pragmatics (2 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Problem domains and programming paradigm • Criteria for good programming language design: – Principles of language design such as orthogonality – Defining control and iteration constructs – Modularization of large software • 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 : [**Stroustrup2013**], [**Josuttis2019**]	

Unit 2: Language Translation and Execution (4 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Execution models for JIT (Just-In-Time), compiler, interpreter • Use of intermediate code, e.g., bytecode • Execution as native code or within a virtual machine • Virtual machines and intermediate languages • Compilation vs. interpretation models	• Explain and understand the differences between compiled, JIT, and interpreted language implementations, including the benefits and limitations of each [Familiarizarse] • Differentiate syntax and parsing from semantics and evaluation [Usar] • Illustrate ambiguity in parsing using nested if-then-else/arithmetic expression and show resolution using precedence order [Evaluar]
Readings : [**Stroustrup2013**], [**Deitel2017**]	

Unit 3: Type Systems Fundamentals (8 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• A type as a set of values together with a set of operations: – Primitive types (e.g., numbers, Booleans) – Compound types built from other types (e.g., records/structs, unions, arrays, lists, functions, references using set operations) • 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 • Declaration models (linking, visibility, scope, lifetime) • Overview of type checking in C++ • Type conversion and casting in C++	• Define and use program pieces (such as functions, classes, methods) that use generic types, including for collections [Usar] • Discuss the differences among generics, subtyping, and overloading [Familiarizarse] • Explain multiple benefits and limitations of static typing in writing, maintaining, and debugging software [Evaluar]
Readings : [**Stroustrup2013**], [**Josuttis2019**]	

Unit 4: Object-Oriented Programming Fundamentals (14 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Definition of classes: fields, methods, and constructors. • Subclasses, inheritance (including multiple inheritance), and method overriding. • Dynamic dispatch: definition of method-call. • Subtyping: – Subtype polymorphism; implicit upcasts in typed languages. – Notion of behavioral replacement: subtypes acting like supertype. – Relationship between subtyping and inheritance. • Classes and objects in C++ • Constructors and destructors • Access specifiers and encapsulation • Inheritance hierarchies • Polymorphism and virtual functions	• Compose a class through design, implementation, and testing to meet behavioral requirements [Usar] • Build a simple class hierarchy utilizing subclassing that allows code to be reused for distinct subclasses [Usar] • Predict and validate control flow in a program using dynamic dispatch [Evaluar] • Design C++ classes with proper encapsulation [Usar]. • Implement inheritance hierarchies for code reuse [Usar].
Readings : [**Stroustrup2013**], [**Josuttis2019**]	

Unit 5: Systems Execution and Memory Model (12 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Direct, indirect, and indexed access to memory location • Run-time representation of data abstractions such as variables, arrays, vectors, records, pointer-based data elements such as linked-lists and trees, and objects • 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 • 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)	• 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 RAII for resource management [Usar]. • Debug memory-related errors in C++ programs [Usar].
Readings : [**Stroustrup2013**], [**Deitel2017**]	

Unit 6: Functional Programming (10 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Lambda expressions and evaluation: – Variable binding and scope rules. – Parameter-passing. – Nested lambda expressions and reduction order. • Using higher-order functions (taking, returning, and storing functions). • Function templates and class templates • Template specialization • Lambda expressions in C++11/14 • Standard Template Library concepts	• Develop useful functions that take and return other functions [Usar] • Correctly interpret variables and lexical scope in a program using function closures [Evaluar] • Use functional encapsulation mechanisms such as closures and modular interfaces [Usar] • Implement generic functions using templates [Usar]. • Use lambda expressions for functional programming [Usar].
Readings : [**Stroustrup2013**], [**Josuttis2019**]	

Unit 7: Object-Oriented Programming: Encapsulation, Subtyping, and Reflection (8 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Collection classes, iterators, and other common library components. • Sequence containers (vector, list, deque) • Associative containers (set, map, unordered_set) • Container adapters (stack, queue, priority_queue) • Iterators and algorithms • Functors and predicates	• 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 [Usar] • Compare and contrast mechanisms to define and protect data elements within procedural, functional, and object-oriented approaches [Evaluar] • Select appropriate STL containers for specific needs [Usar].
Readings : [**Stroustrup2013**], [**Josuttis2019**]	

Unit 8: Advanced Programming Constructs (6 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Object-oriented abstractions: multiple inheritance, mixins, traits, multimethods • Metaprogramming: macros, generative programming, model-based development • Operator overloading • Move semantics and rvalue references • Exception handling in C++ • Namespaces and modular programming	• 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 move semantics for performance optimization [Usar].
Readings : [**Stroustrup2013**], [**Josuttis2019**]	

Unit 9: Shell Scripting (4 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• System commands: – Interface with operating systems • Piping • File abstraction and operators • Programs and processes • Automating compilation and testing of C++ programs through scripts (Makefiles, build/test scripts)	• Create and execute automated scripts to manage various system tasks [Usar]
Readings : [**Robbins2005**]	

7. WORKPLAN

7.1 Methodology

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

7.2 Theory Sessions

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

7.3 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.

8. EVALUATION SYSTEM

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

9. BASIC BIBLIOGRAPHY