

# NATIONAL UNIVERSITY OF SAN MARCOS

FACULTY OF MATHEMATICAL SCIENCES

SYLLABUS

## 1. GENERAL INFORMATION

| | | |
|---|---|---|
| **1.1 Course name** | : | Algorithms and Data Structures |
| **1.2 Code** | : | CS210 |
| **1.3 Academic Program** | : | Scientific Computing |
| **1.4 Credits** | : | 4 |
| **1.5 Prerrequisites** | : | CS113. Objects-oriented programming II. ($3^{rd}$ 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 course provides a comprehensive study of fundamental data structures and algorithms with emphasis on implementation using advanced C++ features. Building on prior C++ knowledge, students will implement generic data structures using templates and traits, ensuring type flexibility and performance. The course covers linear structures (vectors, various linked lists), tree structures (binary, AVL, B-trees), heaps, hash tables, tries, and concurrent data structures. Special focus is placed on modern C++ techniques including variadic templates for generic operations and concurrent programming for thread-safe data structures.

## 4. GOALS

- Implement generic data structures using templates and traits.
- Master advanced tree structures and balanced trees.
- Understand and implement concurrent data structures.
- Apply variadic templates for generic operations on containers.
- Analyze algorithmic complexity of all implemented structures.
- Design efficient data structures for specific problem constraints.

## 5. COMPETENCES

**ABET-1)** An ability to identify, formulate, and solve broadly defined technical or scientific problems by applying knowledge of mathematics and science and/or technical topics to areas relevant to the discipline. (Usage)

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

## 6. TOPICS

| **Unit 1: Advanced Foundation for Data Structures (8 hours)** | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • Template metaprogramming review and advanced traits.<br>• Type traits and SFINAE for conditional compilation.<br>• Variadic templates and parameter packs.<br>• Perfect forwarding and universal references.<br>• Lambda expressions with generic captures.<br>• Policy-based design and CRTP (Curiously Recurring Template Pattern).<br>• Concept-based constraints (C++20 concepts if applicable). | • Design type traits for customizing data structure behavior [Usar].<br>• Implement variadic templates for generic container operations [Usar].<br>• Apply policy-based design to create configurable data structures [Evaluar].<br>• Use perfect forwarding to implement efficient constructors [Usar]. |
| **Readings :** [**Stroustrup2013**], [**Vandevoorde2017**] | |

| **Unit 2: Foundational Data Structures (8 hours)** | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • Abstract Data Type (ADT) and operations on an ADT:<br>  – Dictionary operations (insert, delete, find)<br>• Arrays:<br>  – Numeric vs non-numeric, character strings<br>  – Single (vector) vs multidimensional (matrix)<br>• Records/Structs/Tuples and Objects<br>• Linked lists (for historical reasons):<br>  – Single vs Double and Linear vs Circular<br>• Stacks<br>• Queues and deques:<br>  – Heap-based priority queue<br>• Hash tables/maps:<br>  – Collision resolution and complexity (e.g., probing, chaining, rehash)<br>• Trees:<br>  – Binary, n-ary, and search trees<br>  – Balanced (e.g., AVL, Red-Black, Heap)<br>• Sets | • For each ADT/Data-Structure in this unit:<br>  – Explain its definition, properties, representation(s), and associated ADT operations.<br>  – Explain step-by-step how the ADT operations associated with the data structure transform it.<br>  [Usar]<br>• For each algorithm in this unit explain step-by-step how the algorithm operates [Usar]<br>• Implement vector with customizable allocation policy using templates [Usar].<br>• Design generic linked list nodes with trait-based memory management [Usar].<br>• Compare time/space complexity of different linear structures [Evaluar].<br>• Apply variadic templates to initialize containers with multiple elements [Usar]. |
| **Readings :** [**Cormen2009**], [**Stroustrup2013**] | |

| **Unit 3: Foundational Data Structures (8 hours)** | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • Trees:<br>– Binary, n-ary, and search trees<br>– Balanced (e.g., AVL, Red-Black, Heap)<br>• Binary tree template implementation with iterator support.<br>• Binary search tree with trait-based comparison operators.<br>• Tree traversal algorithms (iterative and recursive).<br>• Expression trees and their applications.<br>• Template-based node structures with configurable payload. | • Given requirements for a problem, develop multiple solutions using various data structures and algorithms. Subsequently, evaluate the suitability, strengths, and weaknesses selecting an approach that best satisfies the requirements [Evaluar]<br>• Explain factors beyond computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data [Familiarizarse]<br>• Implement generic binary tree with configurable traversal order [Usar].<br>• Design expression tree evaluator using visitor pattern [Usar].<br>• Analyze tree operation complexities for different balancing strategies [Evaluar]. |
| **Readings :** [**Cormen2009**], [**Knuth1997**] | |

| **Unit 4: Balanced and Multi-way Trees (8 hours)** | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • AVL tree implementation with template-based balancing trait.<br>• Red-black tree properties and operations.<br>• B-Tree template implementation for disk-based storage.<br>• 2-3 Tree and 2-3-4 Tree structures.<br>• Digital trie (prefix tree) implementation.<br>• Suffix trees and their applications in string processing.<br>• Node splitting/merging strategies for B-trees. | • Implement self-balancing trees using policy-based design [Usar].<br>• Design B-tree for database index applications [Usar].<br>• Evaluate trade-offs between different balanced tree structures [Evaluar].<br>• Implement trie for efficient string prefix matching [Usar]. |
| **Readings :** [**Cormen2009**], [**Knuth1997**], [**Sedgewick2011**] | |

| **Unit 5: Foundational Data Structures (8 hours)** | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • Trees:<br>– Binary, n-ary, and search trees<br>– Balanced (e.g., AVL, Red-Black, Heap)<br>• Binary heap implementation using array representation.<br>• Min-heap and max-heap variants.<br>• Heapify algorithms and complexity analysis.<br>• Priority queue ADT implementation.<br>• Binomial heap and Fibonacci heap overview.<br>• Heap applications: Dijkstra's algorithm, heap sort. | • Explain the heap property and the use of heaps as an implementation of a priority queue [Familiarizarse]<br>• Implement generic heap with configurable comparison trait [Usar].<br>• Design priority queue supporting multiple priority update strategies [Usar].<br>• Analyze heap operations complexity for different heap types [Evaluar].<br>• Apply heap in graph algorithm implementations [Usar]. |
| **Readings :** [**Cormen2009**], [**Sedgewick2011**] | |

| **Unit 6: Foundational Data Structures (8 hours)** | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • Hash tables/maps:<br>– Collision resolution and complexity (e.g., probing, chaining, rehash)<br>• Hash function design and properties.<br>• Separate chaining implementation with linked lists.<br>• Open addressing: linear probing, quadratic probing, double hashing.<br>• Perfect hashing and minimal perfect hashing.<br>• Cuckoo hashing and hopscotch hashing.<br>• Hash table resizing strategies and load factor management. | • Explain how collision avoidance and collision resolution is handled in hash tables [Familiarizarse]<br>• Design generic hash table with configurable collision resolution [Usar].<br>• Implement custom hash functions for user-defined types [Usar].<br>• Evaluate performance of different hashing schemes [Evaluar].<br>• Apply hash tables in compiler symbol table implementation [Usar]. |
| **Readings :** [**Cormen2009**], [**Knuth1997**] | |

| Unit 7: Concurrent Data Structures (8 hours) | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • Thread-safe data structure design principles.<br>• Lock-based concurrent linked lists.<br>• Lock-free and wait-free algorithms.<br>• Concurrent hash tables with fine-grained locking.<br>• Atomic operations and memory ordering.<br>• Concurrent queues (bounded and unbounded).<br>• Concurrent stacks and elimination techniques. | • Design thread-safe data structures using mutex and lock guards [Usar].<br>• Implement lock-free linked list using atomic operations [Usar].<br>• Analyze performance of concurrent vs sequential data structures [Evaluar].<br>• Apply concurrent queues in producer-consumer patterns [Usar]. |
| **Readings :** [**Herlihy2012**], [**Williams2019**] | |

| Unit 8: Specialized Structures and Applications (8 hours) | |
|---|---|
| **Competences Expected: ABET-1,ABET-2** | |
| **Topics** | **Learning Outcomes** |
| • Persistent data structures.<br>• Disjoint-set union (union-find) with path compression.<br>• Bloom filters and their probabilistic properties.<br>• Skip lists for probabilistic balanced search.<br>• Segment trees and Fenwick trees for range queries.<br>• Spatial data structures: k-d trees, quadtrees.<br>• Cache-oblivious data structures. | • Implement persistent data structures with structural sharing [Usar].<br>• Design union-find with both union by rank and path compression [Usar].<br>• Apply specialized structures to solve domain-specific problems [Evaluar].<br>• Evaluate space-time tradeoffs in probabilistic data structures [Evaluar]. |
| **Readings :** [**Okasaki1999**], [**Sedgewick2011**] | |

## 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