San Pablo Catholic University (UCSP)
Undergraduate Program in
Computer Science
SILABO

Universidad Católica San Pablo
2020-I

CS112. Computer Science I (Mandatory)

1. General information

1.1 School	:	Ciencia de la Computación
1.2 Course	:	CS112. Computer Science I
1.3 Semester	:	2^{do} Semestre.
1.4 Prerrequisites	:	CS111. Videogames Programming. (1^{st} Sem)
1.5 Type of course	:	Mandatory
1.6 Learning modality	:	Virtual
1.7 Horas	:	2 HT; 2 HP; 4 HL;
1.8 Credits	:	5

2. Professors

3. Course foundation

This is the second course in the sequence of introductory courses in computer science. The course will introduce students in the various topics of the area of computing such as: Algorithms, Data Structures, Software Engineering, etc.

4. Summary

1. General overwiew of Programming Languages 2. Virtual Machines 3. Basic Type Systems 4. Fundamental Programming Concepts 5. Object-Oriented Programming 6. Algorithms and Design 7. Algorithmic Strategies 8. Basic Analysis 9. Fundamental Data Structures and Algorithms

5. Generales Goals

- Introduce the student to the foundations of the object orientation paradigm, allowing the assimilation of concepts necessary to develop information systems.

6. Contribution to Outcomes

This discipline contributes to the achievement of the following outcomes:

1) Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions. (**Assessment**)

2) Design, implement and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline. (**Assessment**)

5) Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline. (**Familiarity**)

6) Apply computer science theory and software development fundamentals to produce computing-based solutions. (**Usage**)

7. Content

UNIT 1: General overwiew of Programming Languages (1)	
Competences:	
Content	**Generales Goals**
• Brief review of programming paradigms. • Comparison between functional programming and imperative programming. • History of programming languages.	• Discuss the historical context for several programming language paradigms [Familiarity]
Readings: Stroustrup2013, **Deitel17**	

UNIT 2: Virtual Machines (1)	
Competences:	
Content	**Generales Goals**
• The virtual machine concept. • Types of virtualization (including Hardware/Software, OS, Server, Service, Network). • Intermediate languages.	• Explain the concept of virtual memory and how it is realized in hardware and software [Familiarity] • Differentiate emulation and isolation [Familiarity] • Evaluate virtualization trade-offs [Assessment]
Readings: Stroustrup2013, **Deitel17**	

UNIT 3: Basic Type Systems (2)

Competences:

Content	**Generales Goals**

• 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, unions, arrays, lists, functions, references)
• Model statement (link, visibility, scope and life time).
• General view of type checking.
	• For both a primitive and a compound type, informally describe the values that have that type [Familiarity]
	• For a language with a static type system, describe the operations that are forbidden statically, such as passing the wrong type of value to a function or method [Familiarity]
	• Describe examples of program errors detected by a type system [Familiarity]
	• For multiple programming languages, identify program properties checked statically and program properties checked dynamically [Usage]
	• Give an example program that does not type-check in a particular language and yet would have no error if run [Familiarity]
	• Use types and type-error messages to write and debug programs [Usage]
	• Explain how typing rules define the set of operations that are legal for a type [Familiarity]
	• Write down the type rules governing the use of a particular compound type [Usage]
	• Explain why undecidability requires type systems to conservatively approximate program behavior [Familiarity]
	• Define and use program pieces (such as functions, classes, methods) that use generic types, including for collections [Usage]
	• Discuss the differences among generics, subtyping, and overloading [Familiarity]
	• Explain multiple benefits and limitations of static typing in writing, maintaining, and debugging software [Familiarity]

Readings: Stroustrup2013, **Deitel17**

UNIT 4: Fundamental Programming Concepts (6)	
Competences:	
Content	**Generales Goals**
• Basic syntax and semantics of a higher-level language • Variables and primitive data types (e.g., numbers, characters, Booleans) • Expressions and assingments • Simple I/O including file I/O • Conditional and iterative control structures • Functions and parameter passing	• Analyze and explain the behavior of simple programs involving the fundamental programming constructs variables, expressions, assignments, I/O, control constructs, functions, parameter passing, and recursion. [Assessment] • Identify and describe uses of primitive data types [Familiarity] • Write programs that use primitive data types [Usage] • Modify and expand short programs that use standard conditional and iterative control structures and functions [Usage] • Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, the definition of functions, and parameter passing [Usage] • Write a program that uses file I/O to provide persistence across multiple executions [Usage] • Choose appropriate conditional and iteration constructs for a given programming task [Assessment] • Describe the concept of recursion and give examples of its use [Familiarity] • Identify the base case and the general case of a recursively-defined problem [Assessment]
Readings: Stroustrup2013, **Deitel17**	

UNIT 5: Object-Oriented Programming (10)

Competences:

Content	**Generales Goals**

• Object-oriented design
  – Decomposition into objects carrying state and having behavior
  – Class-hierarchy design for modeling
• Object-oriented idioms for encapsulation
  – Privacy and visibility of class members
  – Interfaces revealing only method signatures
  – Abstract base classes
• Definition of classes: fields, methods, and constructors
• Subclasses, inheritance, and method overriding
• Subtyping
  – Subtype polymorphism; implicit upcasts in typed languages
  – Notion of behavioral replacement: subtypes acting like supertypes
  – Relationship between subtyping and inheritance
• Using collection classes, iterators, and other common library components
• Dynamic dispatch: definition of method-call
	• Design and implement a class [Usage]
	• Use subclassing to design simple class hierarchies that allow code to be reused for distinct subclasses [Usage]
	• Correctly reason about control flow in a program using dynamic dispatch [Usage]
	• Compare and contrast (1) the procedural/functional approach—defining a function for each operation with the function body providing a case for each data variant—and (2) the object-oriented approach—defining a class for each data variant with the class definition providing a method for each operation Understand both as defining a matrix of operations and variants [Assessment]
	• Explain the relationship between object-oriented inheritance (code-sharing and overriding) and subtyping (the idea of a subtype being usable in a context that expects the supertype) [Familiarity]
	• Use object-oriented encapsulation mechanisms such as interfaces and private members [Usage]
	• 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 [Usage]

Readings: Stroustrup2013, **Deitel17**

UNIT 6: Algorithms and Design (3)

Competences:

Content	**Generales Goals**
• Problem-solving strategies – Iterative and recursive mathematical functions – Iterative and recursive traversal of data structures – Divide-and-conquer strategies • The role of algorithms in the problem-solving process • Problem-solving strategies – Iterative and recursive mathematical functions – Iterative and recursive traversal of data structures – Divide-and-conquer strategies • Fundamental design concepts and principles – Abstraction – Program decomposition – Encapsulation and information hiding – Separation of behaivor and implementation	• Discuss the importance of algorithms in the problem-solving process [Familiarity] • Discuss how a problem may be solved by multiple algorithms, each with different properties [Familiarity] • Create algorithms for solving simple problems [Usage] • Use a programming language to implement, test, and debug algorithms for solving simple problems [Usage] • Implement, test, and debug simple recursive functions and procedures [Usage] • Determine whether a recursive or iterative solution is most appropriate for a problem [Assessment] • Implement a divide-and-conquer algorithm for solving a problem [Usage] • Apply the techniques of decomposition to break a program into smaller pieces [Usage] • Identify the data components and behaviors of multiple abstract data types [Usage] • Implement a coherent abstract data type, with loose coupling between components and behaviors [Usage] • Identify the relative strengths and weaknesses among multiple designs or implementations for a problem [Assessment]

Readings: Stroustrup2013, **Deitel17**

UNIT 7: Algorithmic Strategies (3)	
Competences:	
Content	**Generales Goals**
• Brute-force algorithms • Greedy algorithms • Divide-and-conquer • Recursive backtracking • Dynamic Programming	• For each of the strategies (brute-force, greedy, divide-and-conquer, recursive backtracking, and dynamic programming), identify a practical example to which it would apply [Familiarity] • Use a greedy approach to solve an appropriate problem and determine if the greedy rule chosen leads to an optimal solution [Assessment] • Use a divide-and-conquer algorithm to solve an appropriate problem [Usage] • Use recursive backtracking to solve a problem such as navigating a maze [Usage] • Use dynamic programming to solve an appropriate problem [Usage] • Determine an appropriate algorithmic approach to a problem [Assessment] • Describe various heuristic problem-solving methods [Familiarity]
Readings: Stroustrup2013, **Deitel17**	

UNIT 8: Basic Analysis (2)	
Competences:	
Content	**Generales Goals**
• Differences among best, expected, and worst case behaviors of an algorithm	• Explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm [Familiarity]
Readings: Stroustrup2013, **Deitel17**	

UNIT 9: Fundamental Data Structures and Algorithms (6)	
Competences:	
Content	**Generales Goals**
• Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max, • Sequential and binary search algorithms • Worst case quadratic sorting algorithms (selection, insertion) • Worst or average case O(N log N) sorting algorithms (quicksort, heapsort, mergesort)	• Implement basic numerical algorithms [Usage] • Implement simple search algorithms and explain the differences in their time complexities [Assessment] • Be able to implement common quadratic and O(N log N) sorting algorithms [Usage] • Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing [Familiarity] • Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data [Familiarity] • Explain how tree balance affects the efficiency of various binary search tree operations [Familiarity] • Demonstrate the ability to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in a particular context [Assessment] • Trace and/or implement a string-matching algorithm [Usage]
Readings: Stroustrup2013, **Deitel17**	

8. Methodology

El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.

El profesor del curso presentará demostraciones para fundamentar clases teóricas.

El profesor y los alumnos realizarán prácticas

Los alumnos deberán asistir a clase habiendo leído lo que el profesor va a presentar. De esta manera se facilitará la comprensión y los estudiantes estarán en mejores condiciones de hacer consultas en clase.

9. Assessment

Continuous Assessment 1 : 20 %

Partial Exam : 30 %

Continuous Assessment 2 : 20 %

Final exam : 30 %