

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



**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 <sup>do</sup> Semestre.
1.4 Prerequisites	:	CS111. Videogames Programming. (1 <sup>st</sup> 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**

**Lecturer**

- Alvaro Henry Mamani-Aliaga <ahmamani@ucsp.edu.pe>
  - PhD in Ciencia de la Computación, UNSA, Perú, 2019.
  - MSc in Ciencia de la Computación, IME-USP, Brasil, 2011.
- Manuel Loaiza Fernandez <meloaza@ucsp.edu.pe>
  - PhD in Informatica, Pontificia Universidad Católica do Rio de Janeiro (PUC-RIO), Brasil, 2009.
  - MSc in Informatica, Pontificia Universidad Católica do Rio de Janeiro (PUC-RIO), Brasil, 2005.

**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 overview 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:

- a) An ability to apply knowledge of mathematics, science. (**Assessment**)
- c) An ability to design a system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability. (**Assessment**)
- h) A recognition of the need for, and an ability to engage in life-long learning. (**Familiarity**)
- i) An ability to use the techniques, skills, and modern computing tools necessary for computing practice. (**Usage**)

## 7. Content

### UNIT 1: General overview of Programming Languages (1)

#### Competences: a

Content	Generales Goals
<ul style="list-style-type: none"><li>• Brief review of programming paradigms.</li><li>• Comparison between functional programming and imperative programming.</li><li>• History of programming languages.</li></ul>	<ul style="list-style-type: none"><li>• Discuss the historical context for several programming language paradigms [Familiarity]</li></ul>

**Readings: Stroustrup2013, Deitel17**

### UNIT 2: Virtual Machines (1)

#### Competences: a

Content	Generales Goals
<ul style="list-style-type: none"><li>• The virtual machine concept.</li><li>• Types of virtualization (including Hardware/Software, OS, Server, Service, Network).</li><li>• Intermediate languages.</li></ul>	<ul style="list-style-type: none"><li>• Explain the concept of virtual memory and how it is realized in hardware and software [Familiarity]</li><li>• Differentiate emulation and isolation [Familiarity]</li><li>• Evaluate virtualization trade-offs [Assessment]</li></ul>

**Readings: Stroustrup2013, Deitel17**

UNIT 3: Basic Type Systems (2)	
Competences: a,i	
Content	Generales Goals
<ul style="list-style-type: none"> <li>• A type as a set of values together with a set of operations <ul style="list-style-type: none"> <li>– Primitive types (e.g., numbers, Booleans)</li> <li>– Compound types built from other types (e.g., records, unions, arrays, lists, functions, references)</li> </ul> </li> <li>• Model statement (link, visibility, scope and life time).</li> <li>• General view of type checking.</li> </ul>	<ul style="list-style-type: none"> <li>• For both a primitive and a compound type, informally describe the values that have that type [Familiarity]</li> <li>• 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]</li> <li>• Describe examples of program errors detected by a type system [Familiarity]</li> <li>• For multiple programming languages, identify program properties checked statically and program properties checked dynamically [Usage]</li> <li>• Give an example program that does not type-check in a particular language and yet would have no error if run [Familiarity]</li> <li>• Use types and type-error messages to write and debug programs [Usage]</li> <li>• Explain how typing rules define the set of operations that are legal for a type [Familiarity]</li> <li>• Write down the type rules governing the use of a particular compound type [Usage]</li> <li>• Explain why undecidability requires type systems to conservatively approximate program behavior [Familiarity]</li> <li>• Define and use program pieces (such as functions, classes, methods) that use generic types, including for collections [Usage]</li> <li>• Discuss the differences among generics, subtyping, and overloading [Familiarity]</li> <li>• Explain multiple benefits and limitations of static typing in writing, maintaining, and debugging software [Familiarity]</li> </ul>
Readings: Stroustrup2013, Deitel17	

UNIT 4: Fundamental Programming Concepts (6)	
Competences: a,i	
Content	Generales Goals
<ul style="list-style-type: none"> <li>• Basic syntax and semantics of a higher-level language</li> <li>• Variables and primitive data types (e.g., numbers, characters, Booleans)</li> <li>• Expressions and assignments</li> <li>• Simple I/O including file I/O</li> <li>• Conditional and iterative control structures</li> <li>• Functions and parameter passing</li> </ul>	<ul style="list-style-type: none"> <li>• 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]</li> <li>• Identify and describe uses of primitive data types [Familiarity]</li> <li>• Write programs that use primitive data types [Usage]</li> <li>• Modify and expand short programs that use standard conditional and iterative control structures and functions [Usage]</li> <li>• 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]</li> <li>• Write a program that uses file I/O to provide persistence across multiple executions [Usage]</li> <li>• Choose appropriate conditional and iteration constructs for a given programming task [Assessment]</li> <li>• Describe the concept of recursion and give examples of its use [Familiarity]</li> <li>• Identify the base case and the general case of a recursively-defined problem [Assessment]</li> </ul>
Readings: Stroustrup2013, Deitel17	

UNIT 5: Object-Oriented Programming (10)	
Competences: a,i	
Content	Generales Goals
<ul style="list-style-type: none"> <li>• Object-oriented design <ul style="list-style-type: none"> <li>– Decomposition into objects carrying state and having behavior</li> <li>– Class-hierarchy design for modeling</li> </ul> </li> <li>• Object-oriented idioms for encapsulation <ul style="list-style-type: none"> <li>– Privacy and visibility of class members</li> <li>– Interfaces revealing only method signatures</li> <li>– Abstract base classes</li> </ul> </li> <li>• Definition of classes: fields, methods, and constructors</li> <li>• Subclasses, inheritance, and method overriding</li> <li>• Subtyping <ul style="list-style-type: none"> <li>– Subtype polymorphism; implicit upcasts in typed languages</li> <li>– Notion of behavioral replacement: subtypes acting like supertypes</li> <li>– Relationship between subtyping and inheritance</li> </ul> </li> <li>• Using collection classes, iterators, and other common library components</li> <li>• Dynamic dispatch: definition of method-call</li> </ul>	<ul style="list-style-type: none"> <li>• Design and implement a class [Usage]</li> <li>• Use subclassing to design simple class hierarchies that allow code to be reused for distinct subclasses [Usage]</li> <li>• Correctly reason about control flow in a program using dynamic dispatch [Usage]</li> <li>• 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]</li> <li>• 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]</li> <li>• Use object-oriented encapsulation mechanisms such as interfaces and private members [Usage]</li> <li>• 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]</li> </ul>
<b>Readings: Stroustrup2013, Deitel17</b>	

<b>UNIT 6: Algorithms and Design (3)</b>	
<b>Competences: a,i</b>	
<b>Content</b>	<b>Generales Goals</b>
<ul style="list-style-type: none"> <li>• Problem-solving strategies <ul style="list-style-type: none"> <li>– Iterative and recursive mathematical functions</li> <li>– Iterative and recursive traversal of data structures</li> <li>– Divide-and-conquer strategies</li> </ul> </li> <li>• The role of algorithms in the problem-solving process</li> <li>• Problem-solving strategies <ul style="list-style-type: none"> <li>– Iterative and recursive mathematical functions</li> <li>– Iterative and recursive traversal of data structures</li> <li>– Divide-and-conquer strategies</li> </ul> </li> <li>• Fundamental design concepts and principles <ul style="list-style-type: none"> <li>– Abstraction</li> <li>– Program decomposition</li> <li>– Encapsulation and information hiding</li> <li>– Separation of behavior and implementation</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Discuss the importance of algorithms in the problem-solving process [Familiarity]</li> <li>• Discuss how a problem may be solved by multiple algorithms, each with different properties [Familiarity]</li> <li>• Create algorithms for solving simple problems [Usage]</li> <li>• Use a programming language to implement, test, and debug algorithms for solving simple problems [Usage]</li> <li>• Implement, test, and debug simple recursive functions and procedures [Usage]</li> <li>• Determine whether a recursive or iterative solution is most appropriate for a problem [Assessment]</li> <li>• Implement a divide-and-conquer algorithm for solving a problem [Usage]</li> <li>• Apply the techniques of decomposition to break a program into smaller pieces [Usage]</li> <li>• Identify the data components and behaviors of multiple abstract data types [Usage]</li> <li>• Implement a coherent abstract data type, with loose coupling between components and behaviors [Usage]</li> <li>• Identify the relative strengths and weaknesses among multiple designs or implementations for a problem [Assessment]</li> </ul>
<b>Readings: Stroustrup2013, Deitel17</b>	

<b>UNIT 7: Algorithmic Strategies (3)</b>	
<b>Competences: a,i</b>	
<b>Content</b>	<b>Generales Goals</b>
<ul style="list-style-type: none"> <li>• Brute-force algorithms</li> <li>• Greedy algorithms</li> <li>• Divide-and-conquer</li> <li>• Recursive backtracking</li> <li>• Dynamic Programming</li> </ul>	<ul style="list-style-type: none"> <li>• 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]</li> <li>• Use a greedy approach to solve an appropriate problem and determine if the greedy rule chosen leads to an optimal solution [Assessment]</li> <li>• Use a divide-and-conquer algorithm to solve an appropriate problem [Usage]</li> <li>• Use recursive backtracking to solve a problem such as navigating a maze [Usage]</li> <li>• Use dynamic programming to solve an appropriate problem [Usage]</li> <li>• Determine an appropriate algorithmic approach to a problem [Assessment]</li> <li>• Describe various heuristic problem-solving methods [Familiarity]</li> </ul>
<b>Readings: Stroustrup2013, Deitel17</b>	

<b>UNIT 8: Basic Analysis (2)</b>	
<b>Competences: a,i</b>	
<b>Content</b>	<b>Generales Goals</b>
<ul style="list-style-type: none"> <li>• Differences among best, expected, and worst case behaviors of an algorithm</li> </ul>	<ul style="list-style-type: none"> <li>• Explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm [Familiarity]</li> </ul>
<b>Readings: Stroustrup2013, Deitel17</b>	

UNIT 9: Fundamental Data Structures and Algorithms (6)	
Competences: a,i	
Content	Generales Goals
<ul style="list-style-type: none"> <li>• Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max,</li> <li>• Sequential and binary search algorithms</li> <li>• Worst case quadratic sorting algorithms (selection, insertion)</li> <li>• Worst or average case <math>O(N \log N)</math> sorting algorithms (quicksort, heapsort, mergesort)</li> </ul>	<ul style="list-style-type: none"> <li>• Implement basic numerical algorithms [Usage]</li> <li>• Implement simple search algorithms and explain the differences in their time complexities [Assessment]</li> <li>• Be able to implement common quadratic and <math>O(N \log N)</math> sorting algorithms [Usage]</li> <li>• Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing [Familiarity]</li> <li>• 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]</li> <li>• Explain how tree balance affects the efficiency of various binary search tree operations [Familiarity]</li> <li>• 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]</li> <li>• Trace and/or implement a string-matching algorithm [Usage]</li> </ul>
<b>Readings: Stroustrup2013, Deitel17</b>	

8. Methodology
<p>El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.</p> <p>El profesor del curso presentará demostraciones para fundamentar clases teóricas.</p> <p>El profesor y los alumnos realizarán prácticas</p> <p>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.</p>

9. Assessment
<p><b>Continuous Assessment 1</b> : 20 %</p> <p><b>Partial Exam</b> : 30 %</p> <p><b>Continuous Assessment 2</b> : 20 %</p> <p><b>Final exam</b> : 30 %</p>