SPC

SOCIEDAD PERUANA DE COMPUTACIÓN

**Peruvian Computing Society (SPC)**
School of Computer Science
Sillabus 2021-I

## 1. COURSE

CS112. Computer Science I (Mandatory)

## 2. GENERAL INFORMATION

| | | |
|---|---|---|
| **2.1 Credits** | : | 5 |
| **2.2 Theory Hours** | : | 2 (Weekly) |
| **2.3 Practice Hours** | : | 4 (Weekly) |
| **2.4 Duration of the period** | : | 16 weeks |
| **2.5 Type of course** | : | Mandatory |
| **2.6 Modality** | : | Face to face |
| **2.7 Prerrequisites** | : | CS111. Computing Foundations. ($1^{st}$ Sem) |

## 3. PROFESSORS

Meetings after coordination with the professor

## 4. INTRODUCTION TO THE COURSE

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.

## 5. GOALS

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

## 6. COMPETENCES

**a)** An ability to apply knowledge of mathematics, science. (**Assessment**)

**b)** An ability to design and conduct experiments, as well as to analyze and interpret data. (**Usage**)

**i)** An ability to use the techniques, skills, and modern computing tools necessary for computing practice. (**Usage**)

## 7. SPECIFIC COMPETENCES

**a10)** Evaluate and apply computational thinking to solve everyday problems ()

**a11)** Efficiently use conditional, repetitive control structures, functions, recursion, sorting and search. ()

**b1)** Identify and efficiently apply various algorithmic strategies and data structures for the solution of a problem given certain space and time constraints. ()

**d1)** Collaborative software development using code repositories and version management (e.g., Git, Bitbucket, SVN) ()

## 8. TOPICS

| Unit 1: General overwiew of Programming Languages (1) | |
|---|---|
| **Competences Expected: a** | |
| **Topics** | **Learning Outcomes** |
| • Brief review of programming paradigms.<br>• Comparison between functional programming and imperative programming.<br>• History of programming languages. | • Discuss the historical context for several programming language paradigms [Familiarity] |
| **Readings :** [Str13], [Dei17] | |

| Unit 2: Virtual Machines (1) | |
|---|---|
| **Competences Expected: a,b** | |
| **Topics** | **Learning Outcomes** |
| • The virtual machine concept.<br>• Types of virtualization (including Hardware/Software, OS, Server, Service, Network).<br>• Intermediate languages. | • Explain the concept of virtual memory and how it is realized in hardware and software [Familiarity]<br>• Differentiate emulation and isolation [Familiarity]<br>• Evaluate virtualization trade-offs [Assessment] |
| **Readings :** [Str13], [Dei17] | |

<table>
<tr><td colspan="2">Unit 3: Basic Type Systems (2)</td></tr>
<tr><td colspan="2">Competences Expected: a,b,i</td></tr>
<tr><td>Topics</td><td>Learning Outcomes</td></tr>
<tr><td>
• A type as a set of values together with a set of operations<br>
  – Primitive types (e.g., numbers, Booleans)<br>
  – Compound types built from other types (e.g., records, unions, arrays, lists, functions, references)<br>
• Model statement (link, visibility, scope and life time).<br>
• General view of type checking.
</td><td>
• For both a primitive and a compound type, informally describe the values that have that type [Familiarity]<br>
• 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]<br>
• Describe examples of program errors detected by a type system [Familiarity]<br>
• For multiple programming languages, identify program properties checked statically and program properties checked dynamically [Usage]<br>
• Give an example program that does not type-check in a particular language and yet would have no error if run [Familiarity]<br>
• Use types and type-error messages to write and debug programs [Usage]<br>
• Explain how typing rules define the set of operations that are legal for a type [Familiarity]<br>
• Write down the type rules governing the use of a particular compound type [Usage]<br>
• Explain why undecidability requires type systems to conservatively approximate program behavior [Familiarity]<br>
• Define and use program pieces (such as functions, classes, methods) that use generic types, including for collections [Usage]<br>
• Discuss the differences among generics, subtyping, and overloading [Familiarity]<br>
• Explain multiple benefits and limitations of static typing in writing, maintaining, and debugging software [Familiarity]
</td></tr>
<tr><td colspan="2">Readings : [Str13], [Dei17]</td></tr>
</table>

| **Unit 4: Fundamental Programming Concepts (6)** | |
|---|---|
| **Competences Expected: a,b,i** | |
| **Topics** | **Learning Outcomes** |
| • Basic syntax and semantics of a higher-level language<br>• Variables and primitive data types (e.g., numbers, characters, Booleans)<br>• Expressions and assingments<br>• Simple I/O including file I/O<br>• Conditional and iterative control structures<br>• 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]<br>• Identify and describe uses of primitive data types [Familiarity]<br>• Write programs that use primitive data types [Usage]<br>• Modify and expand short programs that use standard conditional and iterative control structures and functions [Usage]<br>• 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]<br>• Write a program that uses file I/O to provide persistence across multiple executions [Usage]<br>• Choose appropriate conditional and iteration constructs for a given programming task [Assessment]<br>• Describe the concept of recursion and give examples of its use [Familiarity]<br>• Identify the base case and the general case of a recursively-defined problem [Assessment] |
| **Readings :** [Str13], [Dei17] | |

<table>
<tr><td colspan="2">Unit 5: Object-Oriented Programming (10)</td></tr>
<tr><td colspan="2">Competences Expected: a,b,i</td></tr>
<tr><td>Topics</td><td>Learning Outcomes</td></tr>
<tr><td>
<ul>
<li>Object-oriented design
<ul>
<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>
<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>
<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>
</td><td>
<ul>
<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>
</td></tr>
<tr><td colspan="2">Readings : [Str13], [Dei17]</td></tr>
</table>

| Unit 6: Algorithms and Design (3) | |
|---|---|
| **Competences Expected: a,b,i** | |
| **Topics** | **Learning Outcomes** |
| • Problem-solving strategies<br>– Iterative and recursive mathematical functions<br>– Iterative and recursive traversal of data structures<br>– Divide-and-conquer strategies<br>• The role of algorithms in the problem-solving process<br>• Problem-solving strategies<br>– Iterative and recursive mathematical functions<br>– Iterative and recursive traversal of data structures<br>– Divide-and-conquer strategies<br>• Fundamental design concepts and principles<br>– Abstraction<br>– Program decomposition<br>– Encapsulation and information hiding<br>– Separation of behavior and implementation | • Discuss the importance of algorithms in the problem-solving process [Familiarity]<br>• Discuss how a problem may be solved by multiple algorithms, each with different properties [Familiarity]<br>• Create algorithms for solving simple problems [Usage]<br>• Use a programming language to implement, test, and debug algorithms for solving simple problems [Usage]<br>• Implement, test, and debug simple recursive functions and procedures [Usage]<br>• Determine whether a recursive or iterative solution is most appropriate for a problem [Assessment]<br>• Implement a divide-and-conquer algorithm for solving a problem [Usage]<br>• Apply the techniques of decomposition to break a program into smaller pieces [Usage]<br>• Identify the data components and behaviors of multiple abstract data types [Usage]<br>• Implement a coherent abstract data type, with loose coupling between components and behaviors [Usage]<br>• Identify the relative strengths and weaknesses among multiple designs or implementations for a problem [Assessment] |
| **Readings :** [Str13], [Dei17] | |

| Unit 7: Algorithmic Strategies (3) | |
|---|---|
| **Competences Expected: a,b,i** | |
| **Topics** | **Learning Outcomes** |
| • Brute-force algorithms<br>• Greedy algorithms<br>• Divide-and-conquer<br>• Recursive backtracking<br>• 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]<br>• Use a greedy approach to solve an appropriate problem and determine if the greedy rule chosen leads to an optimal solution [Assessment]<br>• Use a divide-and-conquer algorithm to solve an appropriate problem [Usage]<br>• Use recursive backtracking to solve a problem such as navigating a maze [Usage]<br>• Use dynamic programming to solve an appropriate problem [Usage]<br>• Determine an appropriate algorithmic approach to a problem [Assessment]<br>• Describe various heuristic problem-solving methods [Familiarity] |
| **Readings :** [Str13], [Dei17] | |

| Unit 8: Basic Analysis (2) | |
|---|---|
| **Competences Expected: a,b,i** | |
| **Topics** | **Learning Outcomes** |
| • 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 :** [Str13], [Dei17] | |

| Unit 9: Fundamental Data Structures and Algorithms (6) | |
|---|---|
| **Competences Expected: a,b,i** | |
| **Topics** | **Learning Outcomes** |
| • Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max,<br>• Sequential and binary search algorithms<br>• Worst case quadratic sorting algorithms (selection, insertion)<br>• Worst or average case O(N log N) sorting algorithms (quicksort, heapsort, mergesort) | • Implement basic numerical algorithms [Usage]<br>• Implement simple search algorithms and explain the differences in their time complexities [Assessment]<br>• Be able to implement common quadratic and O(N log N) sorting algorithms [Usage]<br>• Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing [Familiarity]<br>• 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]<br>• Explain how tree balance affects the efficiency of various binary search tree operations [Familiarity]<br>• 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]<br>• Trace and/or implement a string-matching algorithm [Usage] |
| **Readings :** [Str13], [Dei17] | |

## 9. WORKPLAN

### 9.1 Methodology

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

### 9.2 Theory Sessions

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

### 9.3 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.

## 10. EVALUATION SYSTEM

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

## 11. BASIC BIBLIOGRAPHY


[Dei17] Deitel & Deitel. *C++17 - The Complete Guide*. 10th. Pearson, 2017. ISBN: 978-0201734843.

[Str13] Bjarne Stroustrup. *The C++ Programming Language*. 4th. Addison-Wesley, 2013. ISBN: 978-0-321-56384-2.