



University of Engineering and Technology
School of Computer Science
Syllabus of Course – Academic Period 2017-I

1. **Code and Name:** CS112. Programming oriented to Objects I
2. **Credits:** 4
3. **Hours of theory and Lab:** 2 HT; 4 HP;
4. **Professor(s)** Dr. Jose Antonio Fiestas Iquira
 - Dr. Ciencias Naturales, UNIHEIDELBERG, Alemania, 2006.
 - Mag. Física, UNIHEIDELBERG, Alemania, 2002.

Dr. Ernesto Cuadros-Vargas

- Dr. Ciencia de la Computación, ICMC-USP, Brasil, 2004.
- Mag. Ciencia de la Computación, ICMC-USP, Brasil, 1998.

Mg. Teófilo Chambilla Aquino

- Mag. Ciencias, UCHILE, Chile, 2015.

Mg. Jorge Luis Alvarado Revata

- Mag. Gestión de la Investigación, UNIA, España, 2016.

Prof. Carlo Christian Malpartida Anton

- Prof. Ingeniería Eléctrica, PUCP, Perú, 2012.

Meetings after coordination with the professor

5. Bibliography

- [PH13] Deitel. P.J and Deitel. H.M. *C++ How to Program (Early Objects Version)*. Deitel, How to Program. Prentice Hall, 2013. ISBN: 9780133378719. URL: <http://books.google.com.pe/books?id=XIZJNQEACAAJ>.
- [Str13] Bjarne Stroustrup. *The C++ Programming Language*. 4th. Addison-Wesley, 2013. ISBN: 978-0-321-56384-2.

6. Information about the course

- (a) **Brief description about the course** This is the second course in the sequence of introductory courses in computer science. The course will serve as a bridge between the paradigm of the imperative and the object-oriented, the other will introduce participants in the various topics of the area of computing as: algorithms, data structures, Software Engineering, etc.
- (b) **Prerequisites:** CS111. Introducción a la Ciencia de la Computación. (1^{er} Sem)
- (c) **Type of Course:** Mandatory

7. Competences

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

8. Contribution to 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**)

9. Competences (IEEE)

C1. An intellectual understanding and the ability to apply mathematical foundations and computer science theory.⇒
Outcome a

C3. An intellectual understanding of, and an appreciation for, the central role of algorithms and data structures.⇒
Outcome c

CS1. Model and design computer-based systems in a way that demonstrates comprehension of the tradeoff involved in design choices.⇒ **Outcome c**

CS2. Identify and analyze criteria and specifications appropriate to specific problems, and plan strategies for their solution.⇒ **Outcome c**

10. List of topics

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

11. Methodology and Evaluation

Theory Sessions:

The development of the theoretical sessions is focused on the student, through his active participation, solving problems related to the course with the individual contributions and discussing real cases of the industry. The students will develop throughout the course a project of application of the tools received in a company.

Lab Sessions:

Practical sessions are held in the laboratory. Laboratory practices are performed in teams to strengthen their communication. At the beginning of each laboratory the development of the practice is explained and at the end the main conclusions of the activity in group form are highlighted.

Oral Presentations :

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

Reading:

Throughout the course different readings are provided, which are evaluated. The average of the notes in the readings is considered as the mark of a qualified practice. The use of the UTEC Online virtual campus allows each student to access the course information, and interact outside the classroom with the teacher and with the other students.

$$NT=6\%TE1+14\%TE2+16\%P1+24\%P2+4\%(SP1+SP2+SP3+SP4+SP5+SP6+SP7+SP8)$$

Where:

- TE_i = Theory evaluation #i, $1 \leq i \leq 2$
- P_j = Theory evaluation #i, $1 \leq j \leq 2$
- SP_k = Set of problems #i, $1 \leq k \leq 8$

12. Content

Unit 1: General overview of Programming Languages (1)	
Competences Expected: C1	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • Discuss the historical context for several programming language paradigms [Familiarity] 	<ul style="list-style-type: none"> • Brief review of programming paradigms. • Comparison between functional programming and imperative programming. • History of programming languages.
Readings : [Str13], [PH13]	

Unit 2: Virtual Machines (1)	
Competences Expected: C2, CS6	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • 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] 	<ul style="list-style-type: none"> • The virtual machine concept. • Types of virtualization (including Hardware/Software, OS, Server, Service, Network). • Intermediate languages.
Readings : [Str13], [PH13]	

Unit 3: Basic Type Systems (2)	
Competences Expected: C1,C2,CS1	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • 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] 	<ul style="list-style-type: none"> • A type as a set of values together with a set of operations <ul style="list-style-type: none"> – 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.
Readings : [Str13], [PH13]	

Unit 4: Fundamental Programming Concepts (6)	
Competences Expected: C1,C2,CS2	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • 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] 	<ul style="list-style-type: none"> • Basic syntax and semantics of a higher-level language • Variables and primitive data types (e.g., numbers, characters, Booleans) • Expressions and assignments • Simple I/O including file I/O • Conditional and iterative control structures • Functions and parameter passing
Readings : [Str13], [PH13]	

Unit 5: Object-Oriented Programming (10)	
Competences Expected: C2,C24,CS1,CS2	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • 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] 	<ul style="list-style-type: none"> • Object-oriented design <ul style="list-style-type: none"> – Decomposition into objects carrying state and having behavior – Class-hierarchy design for modeling • Object-oriented idioms for encapsulation <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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
Readings : [Str13], [PH13]	

Unit 6: Algorithms and Design (3)	
Competences Expected: C1,C2,C23,CS6	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • 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] 	<ul style="list-style-type: none"> • Problem-solving strategies <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – Iterative and recursive mathematical functions – Iterative and recursive traversal of data structures – Divide-and-conquer strategies • Fundamental design concepts and principles <ul style="list-style-type: none"> – Abstraction – Program decomposition – Encapsulation and information hiding – Separation of behavior and implementation
Readings : [Str13], [PH13]	

Unit 7: Algorithmic Strategies (3)	
Competences Expected: C1,C2,C24,CS1	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • 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] 	<ul style="list-style-type: none"> • Brute-force algorithms • Greedy algorithms • Divide-and-conquer • Recursive backtracking • Dynamic Programming
Readings : [Str13], [PH13]	

Unit 8: Basic Analysis (2)	
Competences Expected: C1,C2,C24,CS1	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • Explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm [Familiarity] 	<ul style="list-style-type: none"> • Differences among best, expected, and worst case behaviors of an algorithm
Readings : [Str13], [PH13]	

Unit 9: Fundamental Data Structures and Algorithms (6)	
Competences Expected: C1,C2,C24,CS1	
Learning Outcomes	Topics
<ul style="list-style-type: none"> • 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] • Describe the implementation of hash tables, including collision avoidance and resolution [Familiarity] • 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] • Solve problems using fundamental graph algorithms, including depth-first and breadth-first search [Usage] • 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] • Describe the heap property and the use of heaps as an implementation of priority queues [Familiarity] • Solve problems using graph algorithms, including single-source and all-pairs shortest paths, and at least one minimum spanning tree algorithm [Usage] • Trace and/or implement a string-matching algorithm [Usage] 	<ul style="list-style-type: none"> • 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) • Hash tables, including strategies for avoiding and resolving collisions • Binary search trees <ul style="list-style-type: none"> – Common operations on binary search trees such as select min, max, insert, delete, iterate over tree • Graphs and graph algorithms <ul style="list-style-type: none"> – Representations of graphs (e.g., adjacency list, adjacency matrix) – Depth- and breadth-first traversals
Readings : [Str13], [PH13]	