University of Engineering and Technology
School of Computer Science
Syllabus of Course
Academic Period 2018-II

1. **Code and Name:** CS1102. Objects oriented programming I
2. **Credits:** 4
3. **Hours of theory and Lab:** 2 HT; 4 HL;
4. **Professor(s)**

Lecturer
- Ernesto Cuadros-Vargas
  - PhD in Computer Science, ICMC-USP, Brazil, 2004.
  - MSc in Computer Science, ICMC-USP, Brazil, 1998.

Practice

- Heider Sanchez
  - PhD in Computer Science, UChile, Chile, 2017.
- José Fiestas
  - PhD in Natural Science, Heidelberg, Germany, 2006.
- Juan Carlos Bueno Villanueva
  - MSc in Information Technology, ESAN, Perú, 2010.
- Maria Hilda Bermejo Rios
  - MSc in Business Administration and Management, Escuela de Postgrado de la Universidad Peruana de Ciencias Aplicadas-UPC, Perú, 2004.
- Isaac Ernesto Bringas
  - MSc in Science, FIIS-UNI, Peru, 2010.
- Jaime Farfán
  - MSc in Information Technology, Universidad de Piura, Perú, 2015.
- Juan Antonio Flores Moroco
  - MSc in Science, Algorithms, Universidad Nacional del Altiplano - Puno, Peru.
- Kenneth Lopez
  - MSc in Computer Science, FGV - Fundação Getúlio Vargas, Brazil, 2017.
- Patricio Morriberón
  - MSc in MBA, Tecnológico de Monterrey, Mexico, 2016.
- Ruben Rivas Medina
  - MSc in Computing, Convenio Pontificia Universidad Católica del Peru y CCL, Peru, 2006.
- Teófilo Chambilla Aquino
  - MSc in Computer Science, Universidad de Chile, Chile, 2016.

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 introduce students in the various topics of the area of computing such as: Algorithms, Data Structures, Software Engineering, etc.

(b) **Prerrequisites:** CS1100. Introduction to Computer Science. (1st Sem)

(c) **Type of Course:** Mandatory

(d) **Modality:** Face to face

7. Specific goals of the Course

- 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**)

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**)

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**)

9. Competences (IEEE)

C1. An intellectual understanding and the ability to apply mathematical foundations and computer science theory.$\Rightarrow$ **Outcome a**

C5. Ability to implement algorithms and data structures in software.$\Rightarrow$ **Outcome b**

CS2. Identify and analyze criteria and specifications appropriate to specific problems, and plan strategies for their solution.$\Rightarrow$ **Outcome b**

CS3. Analyze the extent to which a computer-based system meets the criteria defined for its current use and future development.$\Rightarrow$ **Outcome i**

C1. An intellectual understanding and the ability to apply mathematical foundations and computer science theory.$\Rightarrow$ **Outcome a**

C5. Ability to implement algorithms and data structures in software.$\Rightarrow$ **Outcome b**

CS2. Identify and analyze criteria and specifications appropriate to specific problems, and plan strategies for their solution.$\Rightarrow$ **Outcome b**

CS3. Analyze the extent to which a computer-based system meets the criteria defined for its current use and future development.$\Rightarrow$ **Outcome i**

10. List of topics

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

11. Methodology and Evaluation
Theory Sessions:
The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

Lab Sessions:
In order to verify their competences, several activities including active learning and roleplay will be developed during lab sessions.

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.

Evaluation System:
The final grade NF is obtained through of:

$$NF = 0.25 * ET + 0.10 * DC + 0.05 * SP1 + 0.10 * SP2 + 0.10 * SP3 + 0.10 * SP4 + 0.10 * PY1 + 0.20 * PY2$$

Where:

ET : Theoretical Evaluation (1)

DC : Performance in class (1)

SP : Problem Set (4)

PY : Project (2)

To pass the course you must obtain 11 or more in the final grade NF.

12. Content

Unit 1: General overwiew of Programming Languages (1)	
Competences Expected: C1	
Learning Outcomes	**Topics**
• Discuss the historical context for several programming language paradigms [Familiarity]	• 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**
• 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]	• 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

• 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]
	• 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.

Readings : [Str13], [PH13]

Unit 4: Fundamental Programming Concepts (6)
Competences Expected: C1,C2,CS2

Learning Outcomes	Topics
• 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]	• 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

Readings : [Str13], [PH13]

Unit 5: Object-Oriented Programming (10)	
Competences Expected: C2,C24,CS1,CS2	
Learning Outcomes	**Topics**
- 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]	- 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
Readings : [Str13], [PH13]	

Unit 6: Algorithms and Design (3)	
Competences Expected: C1,C2,C23,CS6	
Learning Outcomes	**Topics**
- 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]	- 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
Readings : [Str13], [PH13]	

Unit 7: Algorithmic Strategies (3)	
Competences Expected: C1,C2,C24,CS1	
Learning Outcomes	**Topics**
- 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]	- 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**
Explain what is meant by "best", "expected", and "worst" case behavior of an algorithm [Familiarity]	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**
- 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]	- 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)
Readings : [Str13], [PH13]	