Introduction to Computer Programming in Engineering and Science

Objectives:	00UV	**Discipline:**	Computer Science, Physics, Mathematics, Biology, Chemistry
Ponderation:	3-2-3	**Course Code:**	360-420-DW
Prerequisite:	00UM, 00UP, 00UT	**Course Credit:**	2 2/3
Corequisite:	00UQ	**Semester:**	4

Introduction

Introduction to Computer Programming in Engineering and Science is offered to students in their final semester. The course focuses on the use of computational methods to solve complex problems in science and engineering and is taught jointly by the Computer Science and Science departments.

The Computer Science part of the course introduces the student to the fundamentals of computer programming (algorithm design, data types and manipulation, control structures, *etc.*) and presents the foundations of the object-oriented approach. The student will learn how to design and implement both numerical and non-numerical (searching and sorting) algorithms in problem solving.

The Science part of the course revisits problems that students have seen in previous science courses but aims to do so in a way that reflects their true complexity. Students will analyze complex problems, derive appropriate models, identify the mathematical equation(s) to be solved, and then create an algorithm that can be implemented as a computer program to obtain a solution. Examples will be taken from a wide range of engineering and scientific fields.

By integrating analysis techniques from computer science, the natural sciences and mathematics and by revisiting concepts from several science disciplines, *Introduction to Computer Programming in Engineering and Science* strongly contributes to the attainment of objective 00UU.

As a science option course, students may complete a comprehensive examination component in this course. In the Science Program, *Introduction to Computer Programming in Engineering and Science* contributes to the following program objectives described in the *Exit Profile* (see table at right).

Goals of the Science Program	Performance Criteria
1. To master the knowledge and skills of a basic scientific education	00UU, 00UV
2. To master the knowledge and skills of a basic general education	
3. To apply the experimental method	3.1, 3.2, 3.4, 3.5
4. To take a systematic approach to problem solving	4.1 – 4.6
5. To use the appropriate data-processing technology	5.2
6. To reason logically	6.1 – 6.4
7. To communicate effectively	7.1, 7.3 – 7.6
8. To learn in an autonomous manner	8.1 – 8.3
9. To work as members of a team	9.1 – 9.3
10. To recognize the links between science, technology and the evolution of society	10.1 – 10.3
11. To construct a personal system of values	11.2, 11.3
12. To identify the context in which scientific ideas originated and evolved	12.1
13. To display attitudes and behaviour compatible with the scientific spirit and method	13.1 – 13.4
14. To apply acquired knowledge and skills to new situations	14.1

Objectives and Standards for *Introduction to Computer Programming in Engineering and Science*

OBJECTIVE	STANDARD	LEARNING OBJECTIVES
Course Objective	**Achievement Context**	
To analyze complex problems in science and engineering and design and implement solutions using computational methods	Based on theoretical situations • Working alone in a time-limited situation • Solving quantitative problems using a computer and relevant software applications as well as software libraries Based on experimental situations • Working alone or in small groups in the laboratory in a time-limited situation • Developing approximate mathematical models and reformulating them in a form suitable for solution by computational methods • Preparing technical reports outlining assumptions, methods and solutions	
	General Performance Criteria	
	1. Proper use of concepts, laws and principles 2. Appropriate use of units and terminology 3. Appropriate use of vector techniques	

OBJECTIVE	STANDARD	LEARNING OBJECTIVES
	4. Correct graphical and mathematical representations of systems 5. Appropriate use of numerical methods 6. Proper justification of methodology 7. Critical assessment of results 8. Recognition of model limitations 9. Submission of technical reports written in clear and concise language	

Elements of Competency	Specific Performance Criteria	Intermediate Learning Objectives
1. To use basic terminology and concepts in computer science	1.1. Precise description of computer hardware	1.1.1. Specify typical computer architecture: CPU, RAM, peripherals including secondary storage devices (e.g., hard disks, DVD drives) 1.1.2. Explain speed metrics (e.g. MHz, GHz) and storage metrics (e.g., K, M, G). 1.1.3. Explain data transfer: input and output, reads and writes.
	1.2. Precise description of computer hardware	1.2.1. Explain the distinction between hardware and software. 1.2.2. Describe the major categories of software. 1.2.3. Describe the nature and importance of operating systems. 1.2.4. Describe the types of language translators: assemblers, compilers and interpreters.
	1.3. Accurate use of the binary number system	1.3.1. Be able to use the binary number system and appreciate its importance in computing.
2. To resolve programming problems	2.1. Correct application of a software development method	2.1.1. Specify the problem requirements. 2.1.2. Analyze the problem. 2.1.3. Develop algorithms and methods. 2.1.4. Generate the test data and expected results. 2.1.5. Verify the program structure. 2.1.6. Implement the program model using a programming language.
	2.2. Correct validation of the program	2.2.1. Test the program, compare the results to the expected results and debug as required.
3. To classify data	3.1. Correct recognition of essential data	3.1.1. Identify given values as well as required input and output. 3.1.2. Determine if the data will be treated as literals, constants or variables. 3.1.3. Explain the meaning of reserved words and identifiers. 3.1.4. Distinguish between integer and floating-point numbers.

OBJECTIVE	STANDARD	LEARNING OBJECTIVES
		3.1.5. Comprehend how data is stored in memory.
		3.1.6. Examine Java's eight primitive data types.
	3.2. Proper classification of data	4.1.1. Distinguish between primitive data type variables and reference type variables.
		4.1.2. Define and initialize variables and use constants.
		4.1.3. Use the assignment operator to change values of variables.
		4.1.4. Use the String class to create String objects.
		4.1.5. Use the string concatenation operator to join strings.
4. To develop code	4.1. Appropriate use of the Java programming language	4.1.1. Explain the steps involved in the program development process.
		4.1.2. Identify programs associated with the processing steps – editor, compiler, and interpreter.
		4.1.3. Use an Integrated Development Environment (IDE) that integrates the software required to develop a computer program..
		4.1.4. Explain Java's two-step execution process and its advantages over other languages.
		4.1.5. Use packages in order to structure the applications.
		4.1.6. Explain how classes are located.
		4.1.7. Describe the purpose of libraries.
		4.1.8. Invoke the print(), println() and printf() methods.
		4.1.9. Supply method arguments.
		4.1.10. Use escape sequences.
		4.1.11. Do GUI I/O using the showInputDialog() and showMessageDialog() methods of the JOptionPane class.
		4.1.12. Do text-based I/O using the methods of the Scanner class.
		4.1.13. Do simple file I/O.
	4.2. Systematic approach to debugging programs	4.2.1. Code the package statement in order to organize the application class.
		4.2.2. Code the import statement in order to import a class from a package.
		4.2.3. Differentiate between logic and syntax errors, compile-time and run-time errors.
		4.2.4. Identify common syntax errors.
		4.2.5. Use the println() method in order to trace program execution.
		4.2.6. Perform unit testing.
	4.3. Properly code a Java program	4.3.1. Apply standard naming conventions.
		4.3.2. Apply recommended indentation and program style techniques.
		4.3.3. Differentiate between internal and external documentation.
		4.3.4. Apply recommended documentation techniques.
	4.4. Proper use of loops	4.4.1. Use a for loop to perform counter-controlled repetition.
		4.4.2. Use a while loop to perform conditional repetition.
		4.4.3. Use a do…while loop to perform conditional repetition, particularly when the outcome is based on user input.
		4.4.4. Determine the most appropriate repetitive structure.
		4.4.5. Outline the function of the break and continue statements.

OBJECTIVE	STANDARD	LEARNING OBJECTIVES	
		4.4.6.	Describe how and why to avoid the use of the break and continue statements.
		4.4.7.	Use loop nesting appropriately.
	4.5. Proper use of arrays	4.5.1.	Declare an array of a primitive data type.
		4.5.2.	Use an initializer list to initialize an array.
		4.5.3.	Use for loops to manipulate data stored in an array.
		4.5.4.	Describe the use of the enhanced for loop to manipulate arrays.
		4.5.5.	"Growing arrays" when they overflow.
		4.5.6.	Define methods that receive entire arrays as parameters and return arrays.
		4.5.7.	Understand and be able to use single and multiple dimension arrays.
	4.6. Effective searching and sorting of data stored in arrays	4.6.1.	Code methods to perform various forms of sequential search on an array and return appropriate indexes and values.
		4.6.2.	Perform searches requiring multiple passes through an array.
		4.6.3.	Perform a sequential search on parallel arrays.
		4.6.4.	Describe the interchange sort algorithm.
		4.6.5.	Apply basic sorting algorithms to sort arrays of primitives.
	4.7. Correct description of object-oriented programming (OOP)	4.7.1.	Define Object-Oriented Programming (OOP).
		4.7.2.	Explain the keystones of programming and OOP – Abstraction, Modularity, Encapsulation, Inheritance and Polymorphism.
	4.8. Proper use of classes	4.8.1.	Model the class in UML (Unified Modeling Language).
		4.8.2.	Explain the structure of a Java class.
		4.8.3.	Identify the class's instance variables (attributes), constructors, methods (behaviors) and prototypes (interfaces).
		4.8.4.	Determine needed access control.
		4.8.5.	Design the algorithms and develop the corresponding methods using a structured design approach.
		4.8.6.	Develop the mainline (application class) algorithm.
		4.8.7.	Identify the method header, method body, declaration statements, method calls, and expressions.
		4.8.8.	Identify the scope, visibility and lifetime of variables: parameters, local variables, instance variables, and static variables.
		4.8.9.	Create an object of a class and understand reference variables, the heap and the stack.
		4.8.10.	Search Java's JDK documentation for information on classes, methods and variables.
	4.9. Solve and implement problems that require decision-making	4.9.1.	Explain the three basic control structures - sequence, selection and repetition.
		4.9.2.	Illustrate control structures using a flowchart.
		4.9.3.	Code simple if/else selection structures.
		4.9.4.	Code nested if/else selection structures.
		4.9.5.	Code selection structures using the switch statement.
		4.9.6.	Evaluate the efficiency of selection structures.
		4.9.7.	Code complex decisions.

OBJECTIVE	STANDARD	LEARNING OBJECTIVES
		4.9.8. Identify the order of precedence of relational and logical operators. 4.9.9. Understand short circuit evaluation. 4.9.10. Understand data validation and its importance to data integrity and system.
	4.10. Design a method that solves a single identifiable task	4.10.1. Solve problems by constructing and using methods that perform one major task. 4.10.2. Create a method. 4.10.3. Invoke a method. 4.10.4. Pass primitive data values to a method. 4.10.5. Return a single value from a method. 4.10.6. Pass a reference value to a method.
5. To apply numerical methods and computer programming techniques to solve scientific and engineering problems	5.1. To solve mathematical problems numerically using Java	5.1.1 Use arithmetic operators respecting the rules of precedence. 5.1.1. Convert arithmetic expressions into Java expressions. 5.1.2. Evaluate Java arithmetic expressions. 5.1.3. Recognize the limitations of numeric data types. 5.1.4. Explain overflow and underflow errors. 5.1.5. Cast from one primitive data type to another. 5.1.6. Explore and use classes from Java's standard packages. 5.1.7. Use methods of the Math class – pow(), sqrt(), round(), random(), min(), and max().
	5.2. Learn appropriate numerical methods to computation of a model	5.2.1. Define appropriate convergence conditions for iterative methods. 5.2.2. Understand and minimize errors related to numerical solution of mathematical equations. 5.2.3. Solve scientific models involving non-linear equations by a variety of methods. 5.2.4. Solve scientific models involving first-order ordinary differential equations (i.e. initial and boundary value problems) using Euler or related methods. 5.2.5. Solve other type of scientific models using the appropriate numerical techniques (e.g. by Fourier or Laplace transform, Monte-Carlo, Newton-Raphson, etc.) 5.2.6. Perform simple numerical optimization given a well-posed problem (e.g. by golden section search method, downhill simplex method, genetic algorithm, etc.)
6. To apply a general approach to model development, application and validation	6.1. Correctly distinguish between laws and models	6.1.1. Perform independent research in the scientific literature as part of the model development process 6.1.2. Develop and apply realistic approximations to complex phenomena 6.1.3. Explain the limitations of such approximations within the context of the model and underlying physical laws 6.1.4. Provide reasonable estimates and justifications for model parameters based on theory, available empirical data, qualitative arguments, direct observations, *etc.* 6.1.5. When possible, assess the accuracy of a model by comparing with theoretical cases or data.

	6.2. Appropriate use of computational methods	6.2.1. Translate model into a form suitable for a computational solution 6.2.2. Identify numerical method(s) best suited to solve a given model 6.2.3. Identify which optimization algorithm is best suited to solve a model 6.2.4. Determine appropriate numerical parameters for solution of the model (*e.g.* sample size, time step, *etc.*) 6.2.5. Evaluate a solution for convergence
	6.3. Evaluation of computational methods	6.3.1. Test the validity of the computational method on simpler problems that are well understood. 6.3.2. Test coded numerical methods for wide range of input to check for possible errors (*e.g.* division by zero, not-a-number (NaN), *etc.*)
	6.4. Analysis of computational results	6.4.1. Identify input required to carry out a calculation and analyze its results 6.4.2. Identify information that should be output by a computation 6.4.3. Judge validity of computational results in view of model approximations and the problem being analyzed 6.4.4. Evaluate the contribution of overall computational errors on the precision of the results 6.4.5. Evaluate the sensitivity of results to various input parameters 6.4.6. Comment on the impact of model approximations on the expected accuracy of model predictions
7. To report computational results in a complete and concise form	7.1. Write a report using a style appropriate to technical writing in English	

Additional Materials

Methodology

- Lecture presentation and participation in class discussions
- Analysis of sample problems and their solutions in class
- Assigned problems
- Laboratory work and reports