Critical Design Review

MISCE project

Mechatronics for Improving and Standardizing Competences in Engineering

Competence: Computer Programming

Experimental platform: Programming exercises

Workgroup: University de Castilla-La Mancha

University of Žilina

This document is the Critical Design Review of the technical competence ‘Computer Programming’. It details the design of the set of programming exercises created.

Version: 1.0

Date: September 4th, 2024

Visit https://misceproject.eu/ for more information.

Index of contents

Index of figures

1 Introduction

1.1 Scope

This document presents the design of the set of programming exercises developed in the framework of MISCE project.

The final objective is to use the develop exercises in the practical lectures of engineering degrees to contribute to the technical competence:

C1. Computer Programming

which related skills are (see Table I):

Table I. Skills of Computer Programming

S1.1.	To be able to use proper data structures
S1.2.	To be able to develop programs to solve engineering problems
S1.3.	To be able to characterize the performance of programs
S1.4.	To be able to debug faulty programs
S1.5.	To be able to optimize programs

1.2 Preliminary definition

¿Objetivo de los distintos tipos de ejercicios?

1.3 Technical requirements

The technical requirements to effectively support the development of the skills listed in Table I through the execution of programming exercises are:

R1. The set of exercises shall include tasks that require the selection and implementation of appropriate data structures (e.g., arrays, lists, stacks, queues, dictionaries) to solve a range of computational and engineering-related problems.

R2. The exercises shall be designed to progressively increase in complexity, enabling students to develop complete programs that address practical engineering scenarios, from basic input/output to advanced problem-solving involving numerical computation, simulation, or data processing.

R3. A subset of exercises shall focus on program performance, encouraging students to measure execution time, memory usage, and algorithmic complexity using built-in or external profiling tools within various programming environments.

R4. Debugging tasks shall be embedded in the exercises, requiring students to analyse, identify, and correct both syntactic and logical errors using debugging tools or manual tracing techniques.

R5. Optimization-oriented exercises shall be included, aimed at refining existing code to improve performance, readability, or resource efficiency by applying algorithmic improvements, refactoring strategies, and data structure tuning.

R6. All exercises shall be compatible with multiple programming environments (such as Python, MATLAB, and C/C++) to support flexibility and reinforce cross-platform coding skills.

R7. The technical setup for the exercises shall ensure accessibility using standard academic resources, including personal laptops or lab computers, without requiring specialized hardware or licensed software beyond freely available tools or academic licenses.

2 Software design

2.1 Preliminaries

Analysing the programming languages used for teaching purpose in the subject which covers computer programming on engineering degrees, we can find the following ones:

- C/C++: C is a low-level, procedural programming language often used to teach fundamental programming concepts and how software interacts with hardware. C++ builds on C by adding object-oriented programming features, making it suitable for more complex applications. Advantages of C/C++ include high performance, fine-grained memory control, and widespread use in embedded systems, real-time applications, and system-level programming. Disadvantages include a steep learning curve, manual memory management that can lead to bugs (e.g., memory leaks or buffer overflows), and a more complex syntax compared to modern languages.

- Python: Python is a high-level, general-purpose programming language widely adopted in engineering education due to its simple and readable syntax. It supports multiple paradigms, including procedural, object-oriented, and functional programming. Advantages include a gentle learning curve, an extensive set of libraries (e.g., NumPy, SciPy, Matplotlib), and strong community support. It's particularly effective for rapid prototyping, data analysis, automation, and artificial intelligence. Disadvantages stem from its interpreted nature, which leads to slower execution compared to compiled languages, and its dynamic typing, which can introduce runtime errors in complex applications.

- Java: Java is an object-oriented, high-level programming language known for its portability and robustness. It compiles to bytecode that runs on the Java Virtual Machine (JVM), making it cross-platform. Java is commonly used in enterprise software, Android development, and web-based systems. **Advantages** include strong typing, automatic memory management (via garbage collection), a vast ecosystem of libraries and frameworks, and widespread industry use. **Disadvantages** include verbose syntax, higher memory usage compared to C/C++, and potentially slower execution due to the abstraction of the JVM.

- MATLAB is a proprietary programming environment developed specifically for numerical computing, widely used in engineering fields such as control systems, signal processing, and simulations. It offers built-in functions and a powerful toolbox ecosystem for matrix operations, data visualization, and algorithm development. Advantages include ease of use for mathematical modeling, integration with Simulink for system simulation, and rapid prototyping capabilities. Disadvantages include licensing costs, limited general-purpose programming capabilities compared to languages like Python or Java, and performance constraints for large-scale or high-performance computing.

Theses 4 environments cover the programming languages used in most of the engineering schools for teaching purposes.

2.2 Example of structure and programming exercises

The programming exercises shall be focused on solving problems that appears in engineering. In this way, the following fields have been established:

1. Introducing programming environment.

Definition: This category focuses on familiarizing students with the basic interface, tools, and workflow of the programming environment they will use (e.g., Python IDLE, MATLAB, an online IDE, etc.). It includes writing and running simple scripts and understanding output.

Example: Write a simple program that prints the message “Welcome to Computer Programming!” on the screen.

2. Type of data and structures.

Definition: This part introduces basic data types (integers, floats, strings, booleans) and composite structures like tuples, lists, dictionaries (in Python) or structs and cell arrays (in MATLAB).

Example: Create variables to store a student’s name, age, and grade point average. Print them using a formatted sentence.

3. Arrays and matrix manipulation.

Definition: Teaches how to validate user input or function arguments and handle potential errors gracefully using conditions or error messages.

Example: Create a function that receives a number and returns its square root. If the input is negative, return an error message instead.

4. Text data manipulation.

Definition: Involves operations on strings and textual data, including concatenation, slicing, searching, replacing, and formatting.

Example: Write a program that takes a sentence and returns the number of vowels and consonants in it.

5. Input Argument and Error Checking Problems.

Definition: Teaches how to validate user input or function arguments and handle potential errors gracefully using conditions or error messages.

Example: Create a function that receives a number and returns its square root. If the input is negative, return an error message instead.

6. Random Number Algorithm Problems.

Definition: Covers the use of random number generators to simulate or solve problems probabilistically, such as dice rolls or Monte Carlo methods.

Example: Simulate rolling two six-sided dice 100 times and display the frequency of each possible sum (from 2 to 12).

7. File Handling Algorithm Problems.

Definition: Focuses on reading from and writing to files, processing file content, and saving results. This includes handling CSV, TXT, or other formats.

Example: Write a program that reads a text file containing numbers (one per line) and calculates their average. Save the result to a new file.

8. Advanced problems.

Definition: These are more complex, integrative challenges that require applying multiple concepts, potentially including algorithm design, recursion, or optimization.

Example: Write a program that implements a basic version of the “Hangman” game, allowing a user to guess letters of a hidden word.

The document Teaching Material contains the detailed list of exercises that can be classified by theme or difficulty level, among others.

References

[1] Downey, A. B. (2015). *Think Python: How to Think Like a Computer Scientist* (2nd ed.). O'Reilly Media.A beginner-friendly introduction to Python programming, widely used in engineering education.

[2] Attaway, S. (2016). *MATLAB: A Practical Introduction to Programming and Problem Solving* (5th ed.). Butterworth-Heinemann. A comprehensive guide to learning MATLAB with a strong focus on engineering problem-solving.

[3] Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley. A definitive reference for C++ programming, especially for high-performance and systems-level applications.

[4] Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall. The foundational text for learning the C language, essential for teaching low-level programming.

[5] Oracle. (2024). *Java Tutorials*. Oracle Corporation. https://docs.oracle.com/javase/tutorial/ The official and up-to-date Java learning resource, useful for teaching object-oriented programming.

[6] Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media. A detailed and comprehensive resource for intermediate and advanced Python programming.

[7] IEEE & ACM. (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf A joint guideline defining core computer science competencies, including programming, applicable to engineering programs.

[8] Guzdial, M., & Ericson, B. (2015). *Introduction to Computing and Programming in Python: A Multimedia Approach* (4th ed.). Pearson. An engaging approach to teaching programming with multimedia applications, ideal for motivating students.

[9] Overland, B. (2020). *Go from MATLAB to Python*. Apress. A practical resource for transitioning from MATLAB to Python in academic and professional contexts.

[10] Hein, D., & Cook, D. (2020). *Practical Programming for Engineers*. Mercury Learning and Information. A hands-on programming guide tailored to the needs and applications of engineering students.