INTERNAL ENGINEERING COMPETITION

RULEBOOK 2025

PROGRAMMING

Table of Contents

What is I.E.C.?

The Ontario Tech Internal Engineering Competition (IEC) is an annual event that challenges undergraduate engineering students to demonstrate their technical, creative, and problem-solving abilities across multiple categories. The IC serves as the qualifying round for the Ontario Engineering Competition (OEC), where winners represent Ontario Tech at the provincial level, with potential advancement to the Canadian Engineering Competition (CEC).

Programming Competition Overview

The Programming Competition challenges teams to design, develop, and implement efficient software solutions to real-world engineering problems under time constraints. Teams will be provided with a challenge and must write programs to solve them accurately, efficiently, and with clear documentation.

Competitors are expected to demonstrate algorithmic thinking, strong coding practices, debugging skills, and logical problem-solving under pressure. The goal is to produce optimized, readable, and functional code that meets the problem requirements and passes performance tests.

Each team consists of up to four competitors. Teams will have three hours to complete the challenge and submit their source code and documentation. Solutions will be judged on accuracy, efficiency, readability, structure, and presentation. The total score will be out of 100 points.

Penalties apply for plagiarism, late submissions, or rule violations.

Key Objectives

- Apply programming and computational thinking to engineering problems.
- Develop optimized, efficient, and reliable code solutions.
- Demonstrate collaboration, time management, and version control.
- Document and communicate logic clearly through comments and reports.
- Highlight coding as an essential engineering skill in modern problem-solving

Rules & Regulations:

1. You can use the internet/Generative AI assistants, but copying code will result in penalties.
2. Any and all publicly used resources should be given proper attribution.
3. You can use any language, package, or library in order to complete the challenge.

Competition Description:

Problem Background

In the classic game of Mastermind, a tray is used where the Mastermind hides a code, and the Guesser has 10 attempts to guess it. The code consists of a sequence of 4 (or sometimes more) pegs, each in a different color. The Guesser makes guesses, which are sequences of 4 (or more) pegs in different colors. A guess is considered 'correct' when the color of each peg in the guess exactly matches the color of the corresponding peg in the Mastermind's code.

- After each guess, the Mastermind provides a score, which includes black and white pegs.
- Black pegs indicate that a guess peg matches both the color and position of a code peg.
- White pegs indicate that a guess peg matches the color of a code peg, but not the position.
- The second objective of the challenge is to display the GAME visually using a GUI.

Task:

Create a function that takes two strings, 'code' and 'guess,' as arguments and returns the score in a dictionary. The 'code' and 'guess' strings consist of numeric digits representing the colors of the pegs. Ensure that no peg is double-scored, and maintain the format of the question. The GUI should be functional and visually appealing, enhancing the gameplay experience.

Example:

guess_score("1423", "5678") → {"black": 0, "white": 0}

guess_score("1423", "2222") → {"black": 1, "white": 0}

guess_score("1423", "1234") → {"black": 1, "white": 3}

guess_score("1423", "2211") → {"black": 0, "white": 2}

Notes:

The code and guess sequences will have the same length.

The "pegs" consist of only digits from 0–9.

Judging Matrix (Rubric)

Category	Criteria	Marks
Code Performance & Result: 50%	Solution is valid, accurate and formatted correctly	/10
	GUI is accurate and functionally useful	/25
	Repeatable results for hidden test cases	/15
Total		**/50**
Overall Strategy & Professionalism: 20%	Code documentation	/4
	Code simplicity and complexity	/6
	Professional code styling	/5
	Appropriate use and organization of packages, libraries, etc.	/5
Design Process	Design Process	/4
	Design justification and defence	/6
	Potential future points of design improvement	/5
Presentation: 30%	Presentation Quality & Visual Aids	/6
	Voice and articulation	/3
	Presentation flow and timing	/3
	Team participation	/3
	Responses to Questions	/3
Total		**/50**
Deduction Total		
Final Total		**/100**

Judging Matrix (Penalties)

Offence	Deduction
Plagiarism	-50 points
Insufficient citation	-50 points
Abstract submitted late	-5 points per minute
Grammar/spelling mistake in abstract	-1 point each
Grammar/spelling in presentation	-1 point per 3 errors
Presentation under 17 minutes	-5 points per minute
Presentation over 20 minutes	-10 points per minute
Absent team member	-25 points