

International Computer Science Competition

Qualification Round Problem Sheet



Problems

ICSC problems consist of conceptual problems and programming problems. The problem type is indicated by the following symbols:

- 💡 Conceptual problem: Requires logical or mathematical reasoning, solveable by hand.
- ⚡ Programming problem: Requires implementing a solution in code.

You can score five points for each problem. The difficulty rating ranges from one star (easiest) to three stars (most challenging).

We have compiled training material to help you with concepts relevant for the posed problems. You can access them here: icscompetition.org/training.

Your Solution

You must provide a written description of your solution for **all problems** in the solution sheet you submit (including code). You may submit handwritten or typed solutions. For programming problems, please also upload your code files. You may submit your coding solutions in Python (3.9), Java (OpenJDK 11), C (C99), or C++ (GCC 15.1). To ensure your code is evaluated correctly, please use the provided code skeleton: icscompetition.org/en/#qualification.

You might want to look at the example solutions for our training problems so you can see what a clear and well-structured submission might look like: https://icscompetition.org/docs/solutions/2025/example/en/ICSC_example_2025.pdf.

Submission Information

When you submit your solution, a participant account will be created automatically to help you manage your submission. You can access your account and submission using the credentials you set during the submission process: <https://icscompetition.org/login>.

Please verify that your solution and code files have uploaded correctly. Any issues with reading or executing your code will be flagged in your participant account. You must submit your solution online by *Sunday, 24 August 2025, 23:59 UTC+0* at <https://icscompetition.org/submit>.

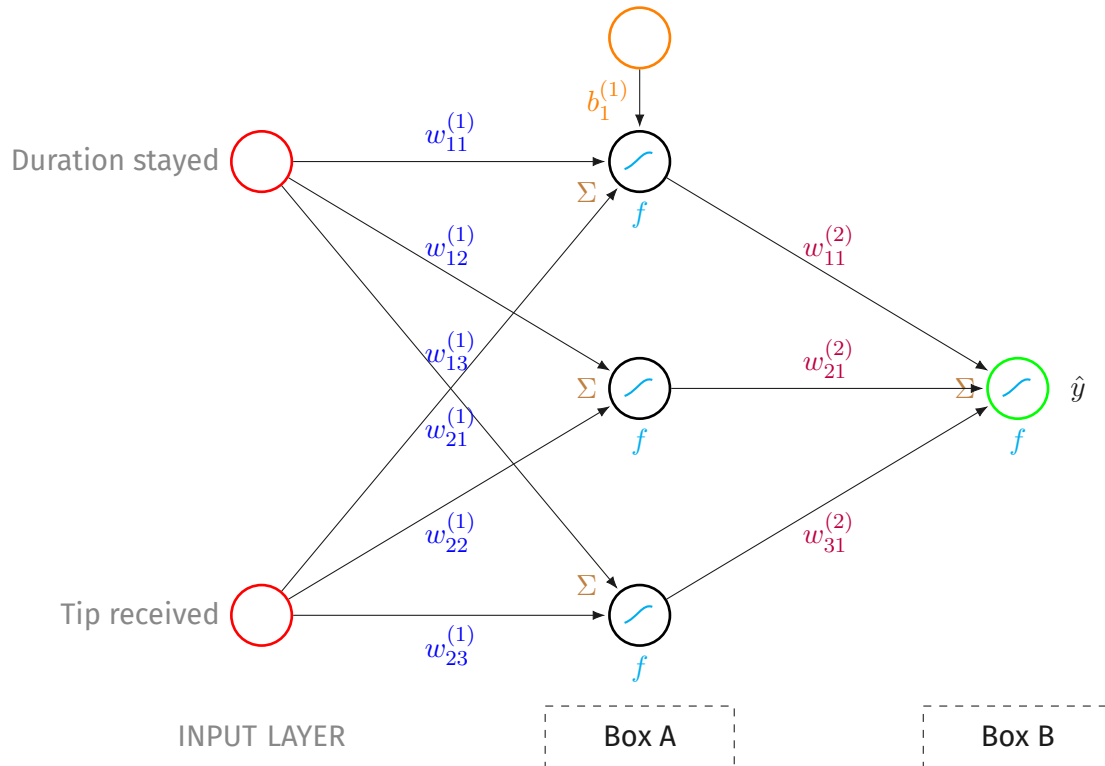
To qualify for the Pre-Final Round, you must score at least 15 points (Junior), 17 points (Youth), or 20 points (Senior). If you have questions or comments, feel free to reach out to us via e-mail at: info@icscompetition.org.

We hope you enjoy tackling the problems. Good luck!

Problem A: Neural Network Components






Consider this illustration of a neural network (Multi-layer Perceptron), that predicts whether a restaurant customer was satisfied with their visit:



Identify and label the following components based on the figure above:

$w_{21}^{(1)}$: Σ : f :

: : :

Box A: Box B: \hat{y} :

</> Problem B: Cake Calculator

A baker has a favorite cake recipe that requires precise amounts of ingredients: **100 units of flour** and **50 units of sugar**. Your task is to help the baker to calculate:

- How many cakes can be made from a given inventory of flour and sugar.
- How much flour and sugar will remain unused after making as many cakes as possible.

Write the following function:

```
list<cake, remaining_flour, remaining_sugar> cake_calculator(flour, sugar).
```

- `flour`: An integer larger 0 specifying the amount of available flour.
- `sugar`: An integer larger 0 specifying the amount of available sugar.
- The function should return a list (or array) of three integers (`cake`, `remaining_flour`, `remaining_sugar`) with the following indices: (0) the number of cakes that can be made, (1) the amount of leftover flour, (2) the amount of leftover sugar.

Hint: There are several valid ways to approach this problem. To help you with the problem, we show you a possible solution below in pseudocode.¹ You may use the pseudocode for your program, or come up with a different solution altogether.

CakeCalculator

Input: Ingredients: flour, sugar

Output: cakeCount, flourLeft, sugarLeft

```

1: flourNeeded ← 100                                /* Recipe requires 100 flour */
2: sugarNeeded ← 50                                  /* Recipe requires 50 sugar */
3: cakeCount ← 0                                     /* Number of cakes made */
4: while true do
5:   if flour < flourNeeded OR sugar < sugarNeeded then
6:     break                                          /* Stop if we can't make any more cakes */
7:   end if
8:   flour ← flour - flourNeeded
9:   sugar ← sugar - sugarNeeded
10:  cakeCount ← cakeCount + 1
11: end while
12: flourLeft ← flour
13: sugarLeft ← sugar
14: return cakeCount, flourLeft, sugarLeft

```

¹Pseudocode is an abstract way of describing algorithms without focusing on the specifics of any particular programming language. It is a useful form of abstraction. Learn how to read pseudocode here: <https://www.geeksforgeeks.org/what-is-pseudocode-a-complete-tutorial/>

💡 Problem C: The School Messaging App



You and your friends have developed a messaging app that operates over the school network. To transmit messages, the app encodes characters into binary sequences (combinations of zeros and ones). However, due to the school's strict data policy, each student is allocated only a small amount of data per day. To make the most of this limited bandwidth, you aim to implement a more efficient encoding strategy.

Information entropy helps determine the theoretical minimum number of bits needed on average to encode messages. For a set of symbols with probabilities p_1, p_2, \dots, p_n , the entropy H is given by:

$$H = - \sum_{i=1}^n p_i \times \log_2(p_i)$$

You know from previous message traffic that characters have the following frequencies:

Character	Probability	Character	Probability
A	0.20	G	0.05
B	0.15	H	0.05
C	0.12	I	0.04
D	0.10	J	0.03
E	0.08	K	0.02
F	0.06	L	0.10

Question 1: Standard text encodings use the same number of bits for each character. Why would using different length codes for characters with different probabilities help you transmit more messages within your data limit? Give an example.

Question 2: Calculate the entropy for the 12-character set above. What does this value represent in terms of optimal encoding?

The Fano method is one way for creating efficient variable-length prefix codes.² You have implemented this algorithm and generated the following code for your character set:

Character	Probability	Fano Code	Character	Probability	Fano Code
A	0.20	000	G	0.05	001
B	0.15	100	H	0.05	1011
C	0.12	010	I	0.04	0111
D	0.10	1100	J	0.03	1101
E	0.08	0110	K	0.02	1111
F	0.06	1010	L	0.10	1110

Question 3: Calculate the average code length of your Fano code and compare it to the theoretical entropy limit. How efficient is your Fano code?

²Details on Fano Codes can be found on: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1999-00/information-theory/fano_codes_4.html

</> Problem D: Word Search Puzzle

A Word Search Puzzle is a common educational game in which players search for given words hidden in a grid of letters. Given a 2D grid of letters, one needs to find all valid words in that grid. Below is an example of such a puzzle containing the words *learning*, *science*, and *fun*:

N	P	K	N	P	I	J	W	J	G
W	L	E	A	R	N	I	N	G	D
V	T	T	S	C	I	E	N	C	E
I	T	Q	T	Y	I	P	C	S	K
B	J	Q	M	A	M	F	U	N	W
M	S	B	P	B	I	O	A	I	X
Q	G	D	D	K	P	C	I	Q	T
C	X	W	N	Z	S	Q	M	C	O
P	I	L	Q	I	P	C	L	Y	X
C	E	X	S	K	I	K	F	G	F

Your task is to implement a function that generates a word search puzzle of **size 10 x 10** based on a list of input words. Specifically write the function:

```
list<list<char>> create_crossword(list<string> words)
```

- `words`: A list of words that should appear in the Word Search Puzzle.
- The function should return a 2D array (or list of lists) of characters representing the puzzle. Each word must appear as a continuous sequence in the grid.
- No specific positioning rules are required. Your words can be arranged horizontally, vertically, or diagonally. Be creative! Exceptional solutions will be awarded.

Hint: You may want to review the following programming concepts before solving this problem: (two-dimensional) arrays, string manipulation, iterations, and nested loops.

💡 Problem E: Functional Completeness of NAND

Logic gates are a fundamental building blocks of computers, as they control how binary information is transmitted through a circuit. A logic gate performs a binary function:

$$f(a, b) \rightarrow \{0, 1\}, \quad \text{where } a, b \in \{0, 1\}.$$

Some gates are particularly important due to a property known as *functional completeness*. A logic gate (or set of gates) is said to be functionally complete if any Boolean function can be constructed using only that gate (or gates).

One such gate is the **NAND** (NOT AND) gate, which outputs 0 only when $a = b = 1$. Thus, it outputs 1 if at least one input is 0.

Prove that the **NAND** gate is functionally complete.

Hint: Can you express the AND, OR, and NOT operations using only NAND?