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 up to 25 points in total. 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 document you submit. Make sure to explain your solution, including any code you have written. You may submit handwritten or typed solutions.

For programming problems, you can optionally also upload your code files to qualify for a special honorary note on your certificate. You may submit your coding solutions in Python (3.9), Java (OpenJDK 11), C (C99), or C++ (GCC 15.1).

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.

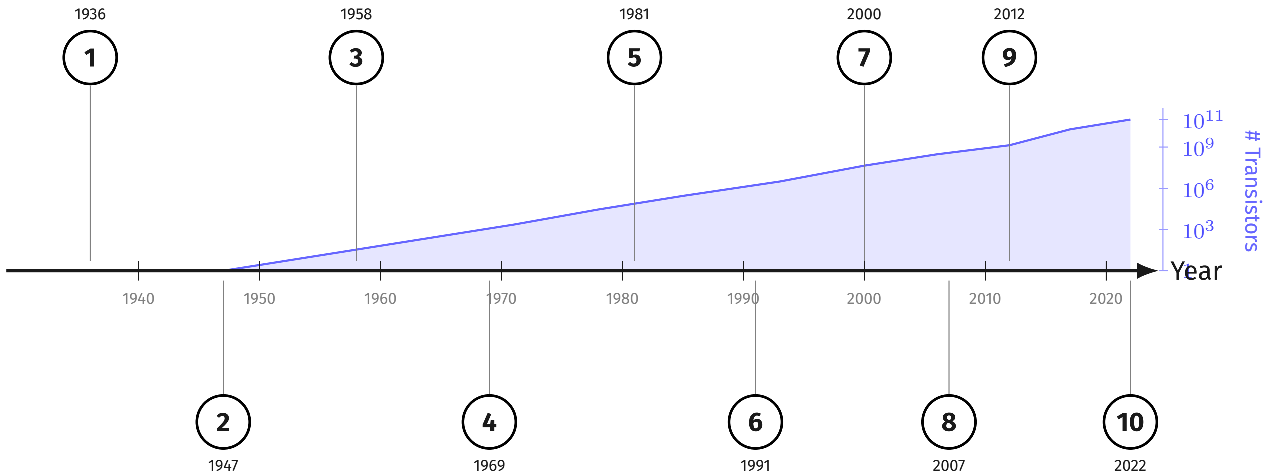
Please verify that your solution and code files have uploaded correctly. Any issues with executing your (optional) code will be flagged in your participant account. You must submit your solution online by *Sunday, 5 July 2026, 23:59 UTC+0* at https://icscompetition.org/submission.

To qualify for the Pre-Final Round, you must score at least 9/13/17 points. 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: A History of Computing (💡, ★☆☆) (5 Points)

The timeline below marks ten landmark events in computing and technology history, plotted against the growth of transistor count on a commercial microprocessor.[1]



Identify and name each numbered event marked on the timeline above (there may be multiple correct answers):

1. (1936):

....

2. (1947):

....

3. (1958):

....

4. (1969):

....

5. (1981):

....

6. (1991):

....

7. (2000):

....

8. (2007):

....

9. (2012):

....

10. (2022):

....

[1] A transistor is a switch or amplifier that regulates the flow of current in a circuit, making it the fundamental building block of all modern computers and electronics.

Problem B: Pixel Art (</>, ★☆☆) (5 Points)

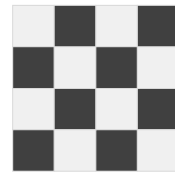
You can download the code skeleton for this problem here.

Pixel art is a form of digital art where images are created by filling individual cells in a grid. Your task is to write a function that generates specific geometric patterns on an $N \times N$ grid, where each cell is either `1` (filled) or `0` (empty).

You must implement the following two shapes:

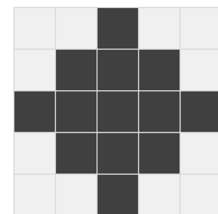
1. Checkerboard (`"checkerboard"`): Cells alternate between filled and empty, like a chessboard. The top-left cell is always `0`. Example for $N = 4$:

```
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
```



2. Diamond (`"diamond"`): A diamond (rhombus) shape centered in the grid. N is always odd for this shape. Example for $N = 5$:

```
0 0 1 0 0
0 1 1 1 0
1 1 1 1 1
0 1 1 1 0
0 0 1 0 0
```



To implement these shapes, write the following function:

```
list<list<int>> generate_shape(int n, string shape)
```

- `n`: The grid size ($N \times N$ grid, $5 \leq N \leq 51$, always odd for the diamond shape).
- `shape`: Either `"checkerboard"` or `"diamond"`.
- The function should return a 2D list/array of integers (`0` or `1`).

Output format: Print the grid as N rows, each containing N space-separated values (`0` or `1`).

Problem C: Moore's Law (💡, ★★☆) (5 Points)

You might have noticed in Problem A that the number of transistors on a commercial microprocessor is not growing linearly. Already in 1975, Gordon Moore stated that the transistor count doubles approximately every **2 years**; a trend that has held for over five decades.

This gives the formula:

$$T(t) = T_0 \times 2^{t/2}$$

where T_0 is the transistor count at a reference year, and t is the number of years elapsed since that reference.

The Intel 4004 was released in 1971 as the world's first commercial single-chip microprocessor, with a transistor count of $2{,}300$.

(a) Imagine yourself in 1971. Using the formula above, predict the transistor count of a microprocessor in 2026.

(b) Compare your prediction to an actual processor from 2026, such as the Apple M5. How well does Moore's Law hold up over 55 years?

(c) Given the actual 2026 transistor count you found in (b), what doubling period (in years) would best fit the growth from the Intel 4004 to today?

Problem D: The Simultaneous Strike (</>, ★★☆) (5 Points)

You can download the code skeleton for this problem here.

You are working on the server for an online 2D fighting game. Two players on separate PCs send their inputs over the internet to a central server every frame (1/60th of a second). The server is the source of truth: it collects both players' actions, updates HP, and broadcasts the result back to both clients.

It is the grand finals. The match ends with Player 1 winning. Player 2 is furious and insists that the finishing blow landed on the exact same frame as Player 1's, and it should have been a double KO. Per the game's rules, all events on the same frame must be applied before checking for a KO. You have a feeling that the code of the function `processGame` has a bug. This is its pseudocode:

processGame

Input: events: list of (player, frame, attack_value) in arbitrary order (due to network jitter)
Output: player1_hp, player2_hp

```
1: stable sort events by frame number
2: for all event in events do
3:     if player1_hp ≤ 0 OR player2_hp ≤ 0 then
4:         break
5:     end if
6:     opponent_hp ← opponent_hp - event.attack_value
7: end for
8: return player1_hp, player2_hp
```

Moreover, the event logs of the fight show:

- `(1, 512, 20)`
- `(2, 512, 20)`

with `player1_hp = 10` and `player2_hp = 15` entering frame 512.

(a) What is wrong with the pseudocode above? How is it causing issues?

(b) Implement the corrected function `processGame` in C, C++, Python, or Java.

```
list<int> processGame(list<tuple<int,int,int>> events, int H)
```

- `events`: A list of tuples `(player, frame, attack_value)` where `player` is 1 or 2, `frame` is a non-negative integer, and `attack_value` is a positive integer.
- `H`: HP for both players before the function call (a positive integer).
- Return a list of two integers: `[hp1, hp2]`, each clamped to a minimum of 0.

Problem E: PageRank in the Age of AI Slop (💡, ★★★) (5 Points)

In 1998, PageRank put Google ahead of every other search engine. The idea: a page is important if important pages link to it. Formally, the PageRank of page p_i is:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where $M(p_i)$ is the set of pages that link *to* p_i, $L(p_j)$ is the number of outbound links from p_j (its out-degree), N is the total number of pages, and $d = 0.85$ is the **damping factor**.

Today, AI-generated "slop" is flooding the web. When could these pages take over the search algorithm?[2]

(a) Explain how PageRank works: the random-surfer interpretation, the role of d, why the equation is recursive, and how it is solved in practice.

(b) Consider N pages split into h human pages and $a = N - h$ adversarial AI pages. Human pages link to k pages chosen uniformly at random from all N pages. AI pages link **only** to other AI pages. Let $\alpha = a/N$.

Derive a closed-form expression for $S_A = \sum_{p \in A} PR(p)$, the total PageRank of all AI pages A, in terms of α and d.

(c) Find the tipping fraction α^* at which $S_A = S_H$ (i.e. AI pages collectively hold as much PageRank as human pages). Then discuss the tipping fraction at $d = 0.85$ and explain issues of PageRank that our scenario reveals.

[2] Google and other modern engines nowadays use hundreds of signals to rank pages. Our fictive scenario focuses on a search engine that uses PageRank exclusively and is thus assuming a simplified scenario.