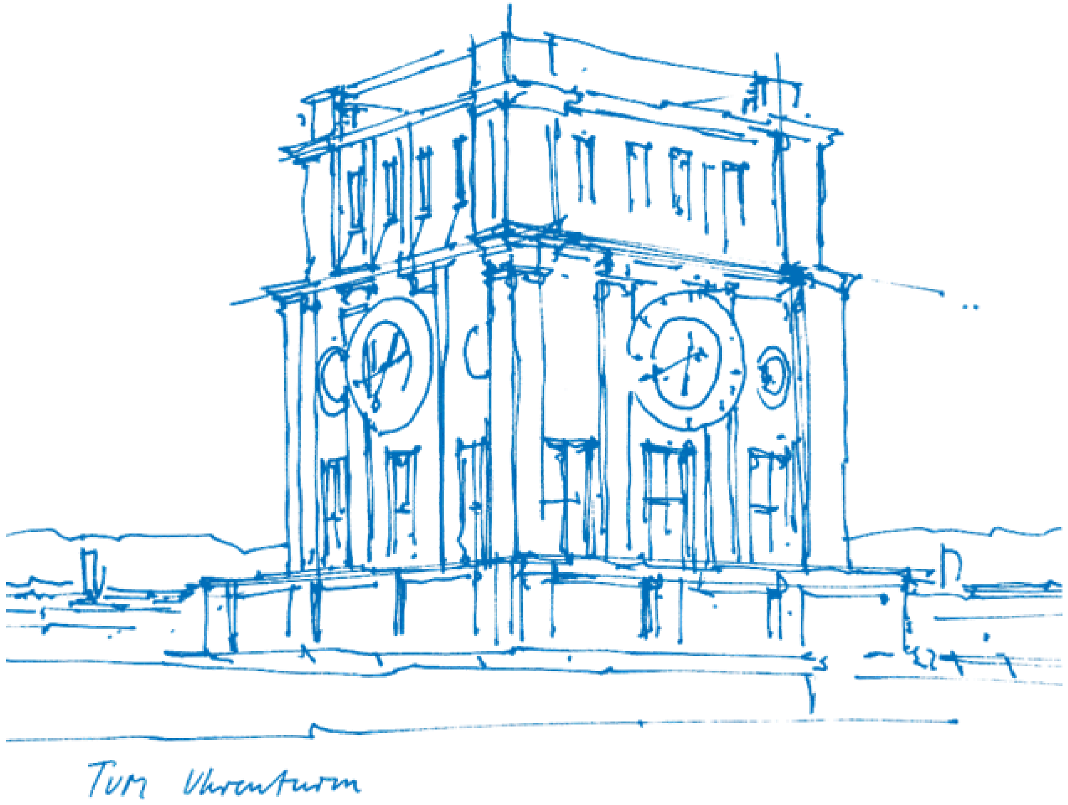
AACPP 2025

Week 1: Basics

Mateusz Gienieczko, Mykola Morozov

School of Computation, Information and Technology
Technical University of Munich

2025.04.29



Structure of the course

Weekly meetings here, Tuesdays, 10:00.

We will focus on explaining solutions to tasks.

Then a short introduction of the new topic and tasks.

Grading

We plan **15 tasks**, each for 10 + 10 points.

Up to two tasks published per week.

First week points count double.

Expect 151 points to pass.

Sign up!

https://nextcloud.in.tum.de/index.php/s/MkipmSfsp3Yws2T

Problem statements

Always read the problem statement carefully.

The **Main section** contains the actual problem to be solved.

Input contains formal input description.

Output contains formal output specification.

Limits set the size parameters of the task.

Evaluation

You upload a single file in `.rs`, `.cpp`, or `.c`.

It gets compiled and ran on **multiple test cases**.

Your program must read from **standard input** and write to **standard output**.

Your output is checked for correctness.

The environment is constraint wrt. capabilities, **time**, and **memory**.

Main section

Translate from author's imagination into a more formal version.

We'll be talking about this extensively next week.

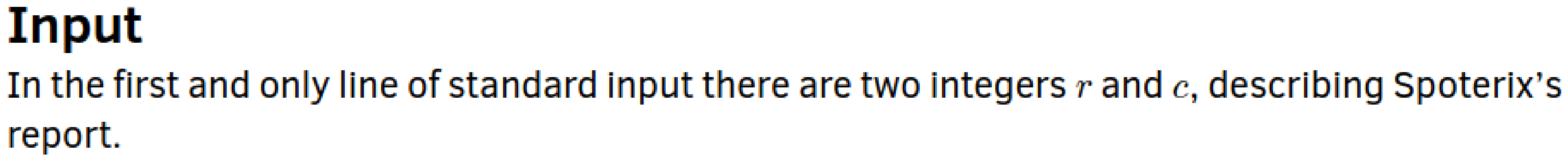
Input

All test cases conform to the formal input description in the test.

Always assume the input is correct (in the sense described in the Input section).

Read from standard input.

Input example (Rust)

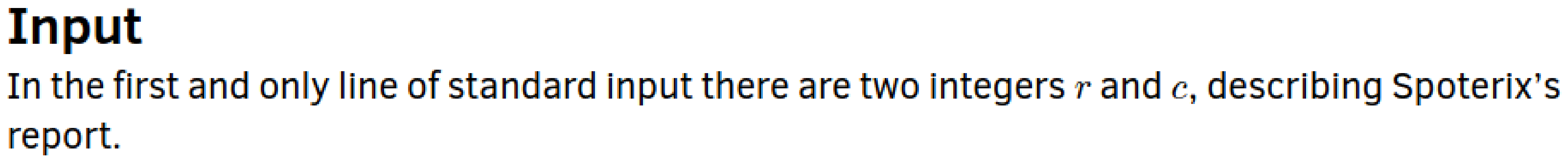


Input

In the first and only line of standard input there are two integers r and c, describing Spoterix's report.

```rust
use std::io::BufRead;
let mut stdin = std::io::stdin().lock().lines();
let line = stdin.next().expect("line")?;
let mut parts = line.split(' ');
let r: u32 = parts.next().expect("r")?.parse()?;
let c: u32 = parts.next().expect("c")?.parse()?;
```

Input example (C++/C)



Input

In the first and only line of standard input there are two integers r and c, describing Spoterix's report.

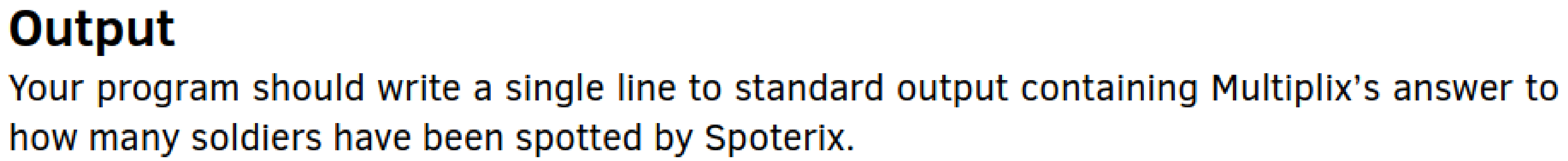
```
uint32_t r, c;
scanf("%d %d", &r, &c);
```

Output

Your output must conform *strictly* to the format.

The only allowed difference is trailing whitespace.

Output example (Rust)

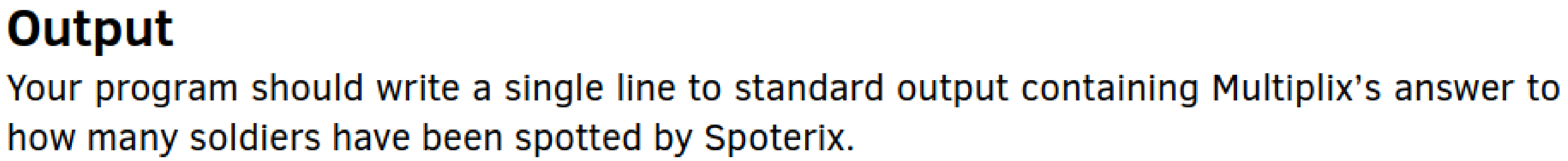


Output

Your program should write a single line to standard output containing Multiplix's answer to how many soldiers have been spotted by Spoterix.

```rust
let result = r * c;
println!("{result}");
```

Output example (C++/C)



Output

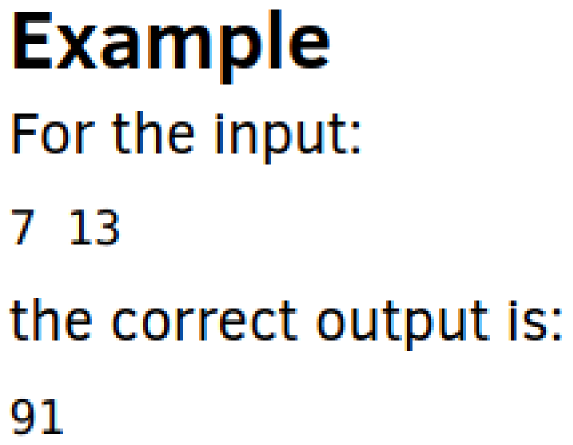
Your program should write a single line to standard output containing Multiplix's answer to how many soldiers have been spotted by Spoterix.

```
uint32_t result = r * c;
printf("%d", result);
```

Examples

Each task contains at least one example of valid input and a correct answer.

They're unlikely to be interesting, they're a smoke test.



Example

For the input:

```
7 13
```

the correct output is:

```
91
```

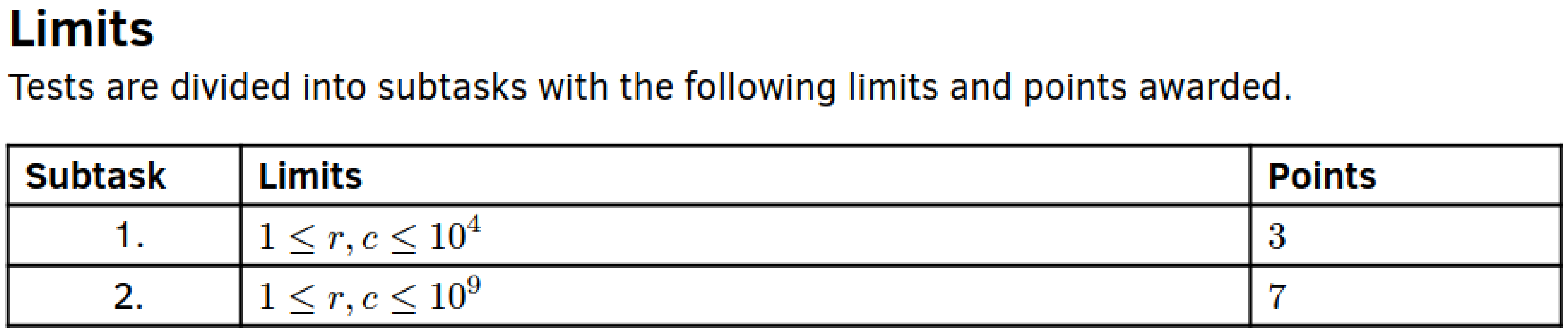
Ok, let’s submit

The system: https://aacpp.db.in.tum.de/c/aacpp-suse-2025.

All submissions happen through the system.

All submissions are automatically judged.

Limits



Limits

Tests are divided into subtasks with the following limits and points awarded.

Subtask	Limits	Points
1.	$1 \le r, c \le 10^4$	3
2.	$1 \le r, c \le 10^9$	7

Limits are very important.

They hint at the complexity of the correct solution.

Scoring

Tests are grouped.

E.g. 1a, 1b, 1c are all in the same group.

You get points for a group if *all* tests in the group are OK.

System constraints – capabilities

Syscalls are heavily restricted.

No filesystem.

No network.

No threading.

Only the standard library.

System constraints – memory

The judging system imposes a strict memory limit, given in the task.

Going above the limit immediately ends execution and results in MLE.

System constraints – time

Execution time is measured with a **virtual clock**.

Instructions are treated as if executing on a 2GHz CPU.

Exceeding the max time results in a TLE.

Estimating resource consumption

Your good friend the $\mathcal{O}$ notation (also o and Ω).

Asymptotic limit on time or memory.

For example, if runtime is $\mathcal{O}(f(n))$ then it performs at most $cf(n)$ operations for any input of size n for some $c \in \mathbb{N}$.

Estimating running time

A 2GHz CPU can technically perform two billion operations per second.

However, even a simple loop adding from 0 to 10^9 takes 3 seconds.

In practice, we're aiming for ≈ 1 to 10 seconds of execution, depending on how complex the operations are.

We choose time limits wrt. our model solutions.

Time limit rules of thumb

There are rules of thumb on what complexity to target.

If we plug max values into the complexity function we should get around 10^8.

For example, for input size n:

- $n \leq 10^6$: target $\mathcal{O}(n)/\mathcal{O}(n \log n)$ ($\log 10^6 < 20$)
-
-
-
-
-

Time limit rules of thumb

There are rules of thumb on what complexity to target.

If we plug max values into the complexity function we should get around 10^8.

For example, for input size n:

- $n \le 10^6$: target $\mathcal{O}(n)/\mathcal{O}(n \log n)$ ($\log 10^6 < 20$)
- $n \le 10^5$: target $\mathcal{O}\left(n \log^2 n\right)/\mathcal{O}\left(n\sqrt{n}\right)$ ($\sqrt{10^5} < 317$)
-
-
-
-

Time limit rules of thumb

There are rules of thumb on what complexity to target.

If we plug max values into the complexity function we should get around 10^8.

For example, for input size n:

- $n \leq 10^6$: target $\mathcal{O}(n)/\mathcal{O}(n \log n)$ ($\log 10^6 < 20$)
- $n \leq 10^5$: target $\mathcal{O}\left(n \log^2 n\right)/\mathcal{O}\left(n\sqrt{n}\right)$ ($\sqrt{10^5} < 317$)
- $n \leq 10^4$: target $\mathcal{O}\left(n^2\right)$
-
-
-

Time limit rules of thumb

There are rules of thumb on what complexity to target.

If we plug max values into the complexity function we should get around 10^8.

For example, for input size n:

- $n \leq 10^6$: target $\mathcal{O}(n)/\mathcal{O}(n \log n)$ $(\log 10^6 < 20)$
- $n \leq 10^5$: target $\mathcal{O}(n \log^2 n)/\mathcal{O}(n\sqrt{n})$ $(\sqrt{10^5} < 317)$
- $n \leq 10^4$: target $\mathcal{O}(n^2)$
- $n \leq 100$: target $\mathcal{O}(n^3)$
-
-

Time limit rules of thumb

There are rules of thumb on what complexity to target.

If we plug max values into the complexity function we should get around 10^8.

For example, for input size n:

- $n \leq 10^6$: target $\mathcal{O}(n)/\mathcal{O}(n \log n)$ ($\log 10^6 < 20$)
- $n \leq 10^5$: target $\mathcal{O}\left(n \log^2 n\right)/\mathcal{O}\left(n\sqrt{n}\right)$ ($\sqrt{10^5} < 317$)
- $n \leq 10^4$: target $\mathcal{O}\left(n^2\right)$
- $n \leq 100$: target $\mathcal{O}\left(n^3\right)$
- $n \leq 20$: target $\mathcal{O}(2^n)$ ($2^{20} \approx 10^6$)
-

Time limit rules of thumb

There are rules of thumb on what complexity to target.

If we plug max values into the complexity function we should get around 10^8.

For example, for input size n:

- $n \leq 10^6$: target $\mathcal{O}(n)/\mathcal{O}(n \log n)$ $(\log 10^6 < 20)$
- $n \leq 10^5$: target $\mathcal{O}\left(n \log^2 n\right)/\mathcal{O}\left(n\sqrt{n}\right)$ $(\sqrt{10^5} < 317)$
- $n \leq 10^4$: target $\mathcal{O}\left(n^2\right)$
- $n \leq 100$: target $\mathcal{O}\left(n^3\right)$
- $n \leq 20$: target $\mathcal{O}(2^n)$ $(2^{20} \approx 10^6)$
- $n \leq 10$: target $\mathcal{O}(n!)$ $(10! \approx 3.6 \cdot 10^6)$

Memory limits

Memory limits are usually easily estimated based on our code.

Allocating an array of n u32 values takes $4n$ bytes.

For example for $n \leq 10^6$ that array takes 4MB.

There’s some overhead from code and stack.

No separate stack limit.

Other formats

Actual competitive programming contests are usually more restricted.

E.g. a round on Codeforces is limited to 120-180 minutes.

Usually no subtasks, penalties for wrong submits (“bombs”).

Team competitive contests where multiple (usually 3) people solve a problemset with **one** computer.

ICPC

International Collegiate Programming Contest

There’s also GCPC.

There was a group on TUM doing ICPC. They sometimes have posters.

icpc@in.tum.de

Other contests

Codeforces

LeetCode

TopCoder

SPOJ

CodeChef

CodinGame

CodeWars

CodeCombat

External resources

Introduction to Algorithms (4th ed.), Cormen et al. (The Algorithms Bible)

Algorithms and Data Structures book by the CS department of Virginia Tech

Competitive Programming Resources by Kunal Kushwaha

Programming Challenges: The Programming Contest Training Manual, Skiena et al.

Competitive Programming book(s) by S. Halim, F. Halim and S. Effendy

Competitive Programming Handbook and others by Antti Laaksonen

Course resources

Official Mattermost

Evaluation system

Course website