

# Lab 12

## Lijnenspel

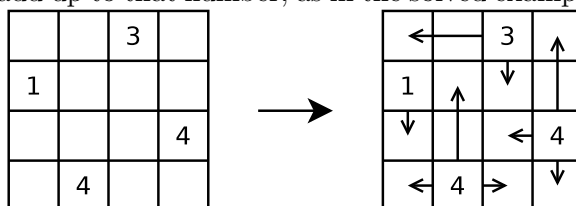
*17 April 2025*

The preview for this lab is given below. It has two semi-separate parts: one to make sure you understand the surface task (a puzzle game called Lijnenspel), and the other part to get you started on using the 2D-vector stuff we've been talking about in class. You can do the preview parts in either order.

Come to lab on Tuesday either with it completed or with a specific written question in your notebook identifying which preview step you got to and what about it you're stuck on.

### Setup 1: Introducing Lijnenspel

Lijnenspel<sup>1</sup> is a puzzle played on a grid (similar to Sudoku or a crossword puzzle) In Lijnenspel, the initial grid contains numbers in some of the squares (as on the left below), and the puzzle solver's job is to draw horizontal and vertical arrows extending out from the numbers to fill the rest of the grid. The total length in squares of all the arrows emanating from a numbered square should add up to that number, as in the solved example on the right:<sup>2</sup>

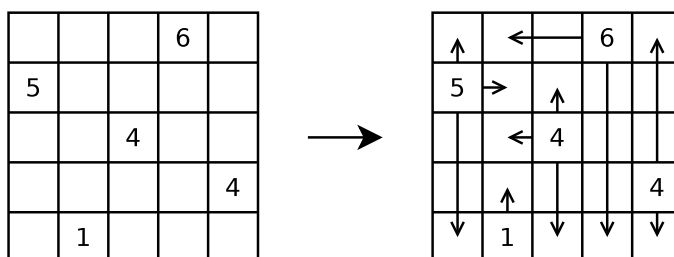


To be a proper Lijnenspel puzzle, the solution should be unique; and indeed any such puzzle with a unique solution is possible to solve without guessing (though the logic may require a bit of work). There are various tactics that can be applied. For instance, if a particular square is only “reachable” from

<sup>1</sup>Also known as “Line Game”, but the authors of the site I pulled examples from are Dutch, and so publish it under both names. “Lijnenspel” looks cooler, no? It’s pronounced “LINE-en-spell”.

<sup>2</sup>This example comes from the description page at <http://www.puzzlepnic.com/genre?lijnenspel>.

one numbered square, there has to be an arrow connecting them; in the following example, the second square in the top row and the fourth square in the bottom row are *only* reachable from the 6—so we could immediately “spend” six to draw arrows from the 6 to those two squares. That makes the right column unreachable except from the 4; and this sort of logic can continue through to complete the puzzle.<sup>3</sup>



To confirm you understand how the Lijnenspel puzzles work, work out at least the first two on the lab worksheet linked from the course page. Bring them with you to lab. Compare notes with other students!

## Setup 2: 2D data

This part is more like the previews you’ve done in the past.

1. Start a program (.cpp file) with a `main` function that reads a single `int` from the user. This will be the size of the Lijnenspel grid (which is square). Make sure this much compiles. (As *always*, but especially this week, try to compile as often as is reasonable, and start running your code as soon as it’s feasible to do so.)
2. Create a 2D vector value that will store the grid. We will represent individual cells in the grid with character values, so this will be a vector of vectors of `char`. Note that it is initially empty (and you can explicitly say so!).
3. Write loops to read in  $n \times n$  characters from `cin` and add them to your grid appropriately. You will almost certainly want this as a nested pair of loops, the outer one working with the whole row and the inner one going character-by-character to build the row.

<sup>3</sup>This puzzle by Zack Butler of RIT, who also provided the inspiration for this lab.

4. To help you debug and understand your code, write a function to take a given grid and print it to `cout`. This is a rare (for me) example of a function where you actually do want to use `cout` inside the function. In your `main`, go ahead and call this function to print out the board that has been read in from the user.
5. Write example files that correspond to either completed or non-completed Lijnenspel puzzles. In a starting-position puzzle, each char will be either a digit from 1 to 9, or a period `'.'` for an open square. In a completed puzzle, all the periods will have been replaced with one of `'<'`, `'>'`, `'^'`, or `'v'`, depending on which direction their arrow was going. (Displaying and storing a length-3 arrow as `">>>"` instead of `"-->"` will make our life easier later, but will still be easy to interpret visually.)
6. Write a function `count_numsquares` that takes a given 2D grid (that is, a `vector<vector<char>>`) and counts and returns how many of the squares in the given grid are number squares. Two notes: first, remember that there is a builtin function to determine whether a character is a digit. Second, in this function you'll need one `for` loop to go through the rows, and another `for` loop *inside* it to go through each element in each row. In your `main` function, after you've printed the grid itself, print how many of its squares are number squares (i.e. the result of this function).
7. (How might you test that function?)