Assignment 2: The Maze

What is recursion

A function that calls itself.

Why recursion isn't great for maze-solving

Recursive functions are especially good at traversing many different paths simultaneously. Since the maze solver only looks for a single path it could easily be done using a loop. For finding the shortest path recursion is very useful as you can traverse many different possible paths at the same time, and pick the best result.

Range-based loop using auto-keyword

```
for(const auto &cnt : a) {readValue = cnt;}
```

When to use const keyword

Const variables are unmutable which means you can't change its value after initial definition. It should be used to define parameters that should not change (like 'pi=3.1415').

Why use the `std::array` type

The `std::array` is a thin wrapper around the standard c array which adds safety and functionality like bound-checking, easy interfacing, pass-by-value and assign, STL type iterators.

Explain issues:

1. To include parts of the standard library, use arrows (`#include <array>`)
2. Use single equal sign for assigning, double for evaluating (`if(a == 1)`)
3. The forloop goes out of bounds and should not include 10'th element.
   (`for (int i = 0; i < 10; ++i)` , `<` instead of `<=`)
4. const elements are unmutable and cannot be redefined

Design Choices

Two different solutions were implemented. Soenke's solution focusses on realism and tries to emulate the right-hand-rule. Tjeerd's solution just tries everything until it finds something that works.
For both solutions the constructor of the Maze class will 1) Create the maze and store it in a 2D array, 2) Find the entrance, 3) Print the maze. The main function will instantiate the Maze class, and then try to solve the maze. We both implemented our own solution to solve the maze.

Tjeerd's Solution

The traverseMaze receives a `position` and a `previousPosition`. If possible (new position != previous position, and new position is not a wall) it will try to move 1 space up (call traverseMaze with the new position), if that's not possible it will try to move 1 space left, then try down, then try right. If none of the options are possible it will mark the current position as "dead end" ('-') and return false. If the position is the endpoint (at the very edge and not starting position) it has solved the maze and will return true.

Soenke's Solution

In real life when you make a turn, your "right" changes from what it was previously. This is emulated in this solution. The player can only move forward and has an orientation. The player will check if there is a wall on the right-hand-side. If there is a wall on his right side the player moves forward. If there is no wall, the player will rotate right and move forward. If there is a wall to his right *and* to his front, the player will rotate left.

This means that at all times the player 'keeps his right hand on the wall', like one would in real life. If a player hits a dead end, it will rotate left twice and then return.