# 1 POU: Day1

```
FUNCTION_BLOCK Day1 IMPLEMENTS IPuzzle
VAR CONSTANT
    LINES               : DINT := 999;
END_VAR
VAR_OUTPUT
    SolutionPart1       : UDINT;
    SolutionPart2       : UDINT;
END_VAR
VAR
    finished            : BOOL := FALSE;
    reader              : LineReader;
    readingSuccess      : BOOL := FALSE;
    leftCollection      : ARRAY[0..LINES] OF DINT;
    rightCollection     : ARRAY[0..LINES] OF DINT;
END_VAR

```

## 1.1 Method: Reset

```
METHOD Reset
VAR
    i : DINT;
END_VAR

```

```
Finished := FALSE;
SolutionPart1 := 0;
SolutionPart2 := 0;

FOR i := 0 TO LINES DO
    leftCollection[i] := 0;
    rightCollection[i] := 0;
END_FOR

```

## 1.2 Method: Solve

```
METHOD Solve
VAR
    lineIndex           : DINT := 0;
    sortedAreaEnd       : DINT;
    tmp                 : DINT;
    tmpLeft             : DINT := 0;
    tmpRight            : DINT := 0;
    i                   : DINT := 0;
END_VAR

```

```
reader(FilePath := 'inputs/day1.txt');
readingSuccess := reader.Done = TRUE AND reader.Error = FALSE;

// -*-* Day 1 *-*-
// Preprocessing, store the data in two sorted lists,
// That makes the later calculations more efficient.
IF readingSuccess = TRUE AND finished = FALSE THEN
    FOR lineIndex := 0 TO LINES DO
        tmpLeft := STRING_TO_DINT(MID(reader.ReadLines[lineIndex], 5, 1));
        tmpRight := STRING_TO_DINT(MID(reader.ReadLines[lineIndex], 5, 9));

```

```
12              // Insert first numbers at the beginning
13              IF lineIndex = 0 THEN
14                  leftCollection[lineIndex] := tmpLeft;
15                  rightCollection[lineIndex] := tmpRight;
16              ELSE // Insert elements while keeping the collections sorted.
17
18                  // Currently inserting a line, so the area is one smaller.
19                  sortedAreaEnd := lineIndex - 1;
20
21                  // Left
22                  ArrayInsertSorted(
23                      Element := tmpLeft,
24                      AreaStart := 0,
25                      AreaEnd := sortedAreaEnd,
26                      TargetArray := leftCollection);
27
28                  // Right
29                  ArrayInsertSorted(
30                      Element := tmpRight,
31                      AreaStart := 0,
32                      AreaEnd := sortedAreaEnd,
33                      TargetArray := rightCollection);
34              END_IF
35          END_FOR
36
37          // Solution part 1: Add up the differences
38          FOR i := 0 TO LINES DO
39              tmp := ABS(leftCollection[i] - rightCollection[i]);
40              SolutionPart1 := SolutionPart1 + DINT_TO_UDINT(tmp);
41          END_FOR
42
43          // Solution part 2: Calculate similarity score
44          tmpRight := 0;
45          FOR tmpLeft := 0 TO LINES DO
46              tmp := 0;
47              FOR i := tmpRight TO LINES DO
48                  // Duplicate found
49                  IF LeftCollection[tmpLeft] = RightCollection[i] THEN
50                      tmp := tmp + 1;
51                  END_IF
52                  // Save score
53                  // The list is sorted, store the end as start for the next line search.
54                  IF LeftCollection[tmpLeft] < RightCollection[i] THEN
55                      tmpRight := i;
56                      SolutionPart2 := SolutionPart2 + DINT_TO_UDINT(LeftCollection[tmpLeft] * tmp)
    ;
57                      EXIT;
58                  END_IF
59              END_FOR
60          END_FOR
61          finished := TRUE;
62      END_IF
63
```

# *1.3 Interface property: IsFinished*

```
1       PROPERTY IsFinished : BOOL
2
```

## 1.3.1 'get' accessor: Get

```
1       IsFinished := finished;
2
```