1 POU: Day1

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

```

1.1 *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 ) ) ;

        // Insert first numbers at the beginning
        IF  lineIndex  =  0  THEN
            leftCollection [ lineIndex ]  :=  tmpLeft ;
            rightCollection [ lineIndex ]  :=  tmpRight ;
        ELSE  // Insert elements while keeping the collections sorted.

            // Currently inserting a line, so the area is one smaller.
            sortedAreaEnd  :=  lineIndex  - 1 ;

            // Left
            ArrayInsertSorted (
                Element  :=  tmpLeft ,
                AreaStart  :=  0 ,
                AreaEnd  :=  sortedAreaEnd ,
                TargetArray  :=  leftCollection ) ;

            // Right
            ArrayInsertSorted (
                Element  :=  tmpRight ,
                AreaStart  :=  0 ,
                AreaEnd  :=  sortedAreaEnd ,
                TargetArray  :=  rightCollection ) ;
        END_IF
    END_FOR

    // Solution part 1: Add up the differences
    FOR  i  :=  0  TO  LINES  DO
        tmp  :=  ABS ( leftCollection [ i ]  -  rightCollection [ i ] ) ;
        SolutionPart1  :=  SolutionPart1  +  DINT_TO_UDINT ( tmp ) ;
    END_FOR

    // Solution part 2: Calculate similarity score
    tmpRight  :=  0 ;
    FOR  tmpLeft  :=  0  TO  LINES  DO
        tmp  :=  0 ;
        FOR  i  :=  tmpRight  TO  LINES  DO
            // Duplicate found
            IF  LeftCollection [ tmpLeft ]  =  RightCollection [ i ]  THEN
                tmp  :=  tmp  +  1 ;
            END_IF
            // Save score
            // The list is sorted, store the end as start for the next line search.
            IF  LeftCollection [ tmpLeft ]  <  RightCollection [ i ]  THEN
                tmpRight  :=  i ;
                SolutionPart2  :=  SolutionPart2  +  DINT_TO_UDINT ( LeftCollection [ tmpLeft ]  *  tmp ) ;
                EXIT ;
            END_IF
        END_FOR
    END_FOR
    Finished  :=  TRUE ;
END_IF

```