1 Interface: IPuzzle

```
INTERFACE IPuzzle

```

1.1 Interface method: Reset

```
METHOD Reset

```

1.2 Interface method: Solve

```
METHOD Solve

```

1.3 Interface property: IsFinished

```
PROPERTY IsFinished : BOOL

```

1.3.1 Interface 'get' accessor: Get

2 Folder: Utility

2.1 POU: ArrayInsertSorted

```
FUNCTION ArrayInsertSorted : BOOL
VAR_INPUT
    Element : DINT;
    AreaStart : DINT;
    AreaEnd : DINT;
END_VAR

VAR_IN_OUT
    TargetArray : ARRAY[*] OF DINT;
END_VAR

VAR_OUTPUT
    ErrorCode : STRING := 'OK';
END_VAR

VAR
    i : DINT;
    arrayStart : DINT := LOWER_BOUND(TargetArray, 1);
    arrayEnd : DINT := UPPER_BOUND(TargetArray, 1);
END_VAR

```

```
IF AreaStart < arrayStart OR AreaEnd > arrayEnd THEN
    ErrorCode := 'Target area outside of array bounds.';
    ArrayInsertSorted := FALSE;
    RETURN;
END_IF

FOR i := AreaStart TO AreaEnd DO
    // Insert before the first larger element
    IF TargetArray[i] > Element THEN
        ShiftArray(
            Direction := ShiftDirection.RIGHT,
```

```
12                  ShiftAmount := 1,
13                  Start := i,
14                  End := AreaEnd,
15                  TargetArray := TargetArray);
16
17              TargetArray[i] := Element;
18              ArrayInsertSorted := TRUE;
19              RETURN;
20          END_IF
21      END_FOR
22
23      // All elements are smaller, append at the end.
24      TargetArray[AreaEnd + 1] := Element;
25      ArrayInsertSorted := TRUE;
26      RETURN;
27
```

2.2 POU: LineReader

```
1       FUNCTION_BLOCK LineReader
2       VAR CONSTANT
3           MAX_LINES : __XWORD := 999;
4           MAX_LINE_LENGTH : INT := 255;
5       END_VAR
6       VAR_INPUT
7           FilePath : STRING;
8       END_VAR
9       VAR_OUTPUT
10          Done : BOOL := FALSE;
11          Error : BOOL := FALSE;
12          ErrorCode : STRING := '';
13          ReadLines : ARRAY[0..MAX_LINES] OF STRING(MAX_LINE_LENGTH);
14          LineCount : WORD := 0;
15      END_VAR
16      VAR
17          CurrentLine : STRING(MAX_LINE_LENGTH) := '';
18          FileHandle : SysFile.RTS_IEC_HANDLE;
19          BytesRead : __XWORD := 0;
20          CharBuffer : STRING(1);
21          pResult : POINTER TO SysFile.RTS_IEC_RESULT;
22          i : __XWORD := 0;
23          Initialized : BOOL := FALSE;
24      END_VAR
25
```

```
1       IF Initialized = FALSE THEN
2           FOR i := 0 TO MAX_LINES DO
3               ReadLines[i] := 'NOT WRITTEN TO';
4           END_FOR
5           Initialized := TRUE;
6       END_IF
7
8       FileHandle := SysFileOpen(szFile := FilePath, am := SysFile.AM_READ, pResult := pResult);
9
10      IF FileHandle = SysFile.RTS_INVALID_HANDLE THEN //File not found
11          ErrorCode := 'File not found, is it on the device?';
12          Error := TRUE;
13          Done := TRUE;
14      ELSE // File open
15          WHILE NOT Done DO
16              BytesRead := SysFileRead(hFile := FileHandle, pbyBuffer := ADR(CharBuffer), ulSize := 1,
        pResult := pResult);
17
18              IF BytesRead > 0 THEN
19                  // A little bit of cheating, only read files with unix line endings
20                  IF CharBuffer = '$N' THEN
```

```
                    IF LineCount <= MAX_LINES THEN
                        ReadLines [ LineCount ] := CurrentLine ;
                        LineCount := LineCount + 1 ;
                        CurrentLine := '' ;
                    ELSE
                        ErrorCode := 'Input file is larger than max line count.' ;
                        Error := TRUE ;
                        Done := TRUE ;
                    END_IF

                ELSE
                    IF LEN ( CurrentLine ) <= MAX_LINE_LENGTH THEN
                        CurrentLine := CONCAT ( CurrentLine , CharBuffer ) ;
                    ELSE
                        ErrorCode := 'Line exceedes max line length' ;
                        Error := TRUE ;
                        Done := TRUE ;
                    END_IF
                END_IF
            ELSE
                Done := TRUE ;
                EXIT ;
            END_IF
        END_WHILE ;
    END_IF

    SysFileClose ( FileHandle ) ;

```

2.3 POU: LineToNumbers

```
    FUNCTION LineToNumbers : INT
    VAR_INPUT
        line            : STRING ;
        separator       : STRING ;
    END_VAR
    VAR_IN_OUT
        numberBuffer    : ARRAY [ * ] OF DINT ;
    END_VAR
    VAR_OUTPUT
        numberCount     : INT           := 0 ;
    END_VAR
    VAR
        separatorLength : INT           := LEN ( separator ) ;
        bufferStart     : DINT              := LOWER_BOUND ( numberBuffer , 1 ) ;
        bufferEnd       : DINT              := UPPER_BOUND ( numberBuffer , 1 ) ;
        bufferIndex     : INT           := 0 ;
        endReached      : BOOL              := FALSE ;
        separatorIndex  : INT           := 0 ;
        tmp             : STRING        := '' ;
        startRight      : INT           := 0 ;
        emergencyBreak  : INT           := 0 ;
    END_VAR

```

```
    WHILE endReached = FALSE DO
        // Just in case something goes wrong,
        // endless loops are annoying.
        emergencyBreak := emergencyBreak + 1 ;
        IF emergencyBreak > 100 THEN
            LineToNumbers := -1 ;
            RETURN ;
        END_IF

        separatorIndex := FIND ( line , separator ) ;
        IF separatorIndex > 0 THEN
```

```
            tmp := LEFT(line, separatorIndex - 1);
            startRight := LEN(line) - (separatorIndex + separatorLength - 1);
            line := RIGHT(line, startRight);
        ELSE
            tmp := line;
            endReached := TRUE;
        END_IF

        bufferIndex := DINT_TO_INT(bufferStart + numberCount);
        IF bufferIndex <= bufferEnd THEN
            numberBuffer[bufferIndex] := STRING_TO_INT(tmp);
            numberCount := numberCount + 1;
        ELSE
            LineToNumbers := -2;
            RETURN;
        END_IF
    END_WHILE

    LineToNumbers := numberCount;

```

2.4 POU: ShiftArray

```
FUNCTION ShiftArray : BOOL
VAR_INPUT
    Direction   : ShiftDirection;
    ShiftAmount : DINT;
    Start       : DINT;
    End         : DINT;
END_VAR
VAR_IN_OUT
    TargetArray : ARRAY[*] OF DINT;
END_VAR
VAR_OUTPUT
    ErrorCode   : STRING := 'OK';
END_VAR
VAR
    i : DINT;
    arrayStart  : DINT := LOWER_BOUND(TargetArray, 1);
    arrayEnd    : DINT := UPPER_BOUND(TargetArray, 1);
    target      : DINT;
END_VAR

```

```
    IF Start < arrayStart OR End > arrayEnd THEN
        ErrorCode := 'Shift start/end outside of array bounds';
        ShiftArray := FALSE;
        RETURN;
    END_IF

    // Shift logic: Always start at the outer bound to prevent overriding values.
    IF Direction = ShiftDirection.RIGHT THEN
        FOR i := End TO Start BY -1 DO
            target := i + ShiftAmount;
            IF target <= arrayEnd AND target >= arrayStart THEN
                TargetArray[target] := TargetArray[i];
            END_IF
        END_FOR
    END_IF

    IF Direction = ShiftDirection.LEFT THEN
        FOR i := Start TO End BY +1 DO
            target := i - ShiftAmount;
            IF target <= arrayEnd AND target >= arrayStart THEN
                TargetArray[target] := TargetArray[i];
            END_IF
```

```
        END_FOR
    END_IF

    ShiftArray := TRUE;
    RETURN;

```

2.5 POU: SIGN

```
    FUNCTION SIGN : INT
    VAR_INPUT
        number : DINT;
    END_VAR

```

```
    IF number > 0 THEN
        SIGN := 1;
    ELSIF number < 0 THEN
        SIGN := -1;
    ELSE
        SIGN := 0;
    END_IF

```

3 POU: PLC_PRG

```
    PROGRAM PLC_PRG
    VAR CONSTANT
        PUZZLE_INDEX        : INT := 1; // last index in array
    END_VAR
    VAR_OUTPUT
        Decoration          : BOOL := TRUE;
    END_VAR
    VAR
        day1                : Day1;
        day2                : Day2;
        puzzles             : ARRAY[0..PUZZLE_INDEX] OF IPuzzle := [day1, day2];
        puzzlesEnabled      : ARRAY[0..PUZZLE_INDEX] OF BOOL;
        puzzleResetTriggers : ARRAY[0..PUZZLE_INDEX] OF F_TRIG;
        i                   : DINT := 0;
    END_VAR

```

```
    // File not found? Put the input on the device with:
    // Double click Device-> Files-> into the PlcLogic/inputs folder

    // Reset puzzle
    FOR i := 0 TO PUZZLE_INDEX DO
        puzzleResetTriggers[i](CLK := puzzlesEnabled[i]);
        IF puzzleResetTriggers[i].Q = TRUE THEN
            puzzles[i].Reset();
        END_IF
    END_FOR

    // Solve puzzle
    FOR i := 0 TO PUZZLE_INDEX DO
        IF puzzlesEnabled[i] = TRUE AND puzzles[i].IsFinished = FALSE THEN
            puzzles[i].Solve();
        END_IF
    END_FOR

```