PS/AA/SvdM/pd 19 May 1980

Programme LISTER (AA Version)

S. van der Meer

This programme will list NODAL files. It will repeatedly ask for a filename and will then list all contents of the file. The programme at present only works under name RT. Any file to be listed must be preceded by its own user prefix if this is not RT.

The listing will consist of

a) date and filename
b) NODAL text if any (LIST output)
c) LISV output
d) a list of all simple string variables saved in the file and their contents (if any)
e) for each array saved in the file, a table giving the contents of each element. For two-dimensional arrays, this is printed in matrix form.

Strings will be printed between quotes. "Control characters" (i.e. those with values i outside the range from 32 to 126 decimal) will be printed as \i. For instance, a strong A defined with

```
$SET A = \9\41\4"VARIABLE"\12\0
```

will be listed as

```
\9\41\4"VARIABLE"\12\0
```

Note that the sequence \41 is printed as \41 and not as ")" as would be expected from the definition above. This will happen for all sequences following \9 or \11, since these will usually represent column or line numbers for displays.

The programme uses a scratch file (of type "symbolic") whose name is (RT)LIST1.. where .. represents the terminal's ODEV, increased by 9. Thus, listings made at the same time from different terminals connected to the same computer will not interfere.

The listing is made both on the terminal screen and on any printer connected in parallel with it.

All internal variable names used in the programme have names ending in .987. This is to avoid conflicts with variables in the files to be listed; it is assumed that no such names occur in these files. In the following description, this suffix will be omitted.

1.10	All files are closed (this may be necessary if a preceding LISTER run was interrupted by ESC). The present ODEV is saved.
1.20	The sequence \19 will put the printer "off line", so that the following input of the filename will not appear there. The name of the file to be listed is kept in STRARG. If no more files should be listed, the RETURN key is pressed.
1.30	Then, ODEV is restored, \17 activates the printer and two pages are ejected.
1.70	If the filename is not "null", the scratch file is opened for writing.
1.80	Line 1.82 will load the file and initiate the listing. If a wrong (non-existing) filename was given, line 1.84 will recover.
1.82	Variable T2 is predefined and saved with the programme. Its contents are executed by the nested $DO commands. The $DO is needed because line 1.82 may be overwritten by a line with the same number from the file to be listed. The nesting is needed to obtain sufficient space for all commands between ERASE ALL and GO 1.86.
T2	First, the LISTER text and variables are erased, then the file to be listed is loaded. T3 contains further commands.
T3	The line printer is activated with \17 and will start on a new page (\12). The date is printed, followed by the filename in the top right corner. The WAIT command is needed with the present printers used at AA, which otherwise may lose lines at this point because of the long time taken for the "page eject". Back to T2.
T2	The LIST output now goes to the normal ODEV (i.e. terminal + printer), followed again by a WAIT to let the printer recover. Then, the LISV output follows, first to screen and printer, then to the scratch file. Finally, groups 1 and 2 are erased.
1.82	The other groups, corresponding with groups in LISTER, are also erased to avoid interference when LISTER is loaded again. Note that the same effect might be obtained with ERASE ALLP; this command is, however, not yet available in the present PS NODAL version. After this line, the text buffer will contain the LISTER text, as well as perhaps some (irrelevant) groups from the file to be listed. All variables from the file to be listed are still present, as well as the LISTER variables.
1.86	The original ODEV is recovered.
1.90	The scratch file, containing the LISV output, is closed for writing and opened for reading. Group 2 will analyse its contents line by line.

2.10-2.16	Initialization, explained later.
2.17	LM contains the maximum number of characters allowed on a printer line. If typing the last character on a line will automatically result in a line feed, LM should not include this last character.
2.20	A line is read and put into T. If T begins with a number rather than a letter, control goes to 3.05. This means that the last LISV line has been reached; all other lines begin with a variable name.
2.30	The variable name (with trailing blanks removed) is stored in VN.
2.40	If this name represents a defined function, line 2.42 is done.
2.42	M was initialized to zero in 2.1. It will contain the number of defined functions in the file. D is a string array containing their names; TT will contain the entire corresponding LISV line, which will later be printed above the listing of the defined function. Return to 2.2 for reading the next LISV line.
2.44	Reached from 2.4 if the variable was not a defined function. W will contain

VAR for a simple string variable
ARR for a string array
INT for an integer array
FLO for a floating array.

2.46	O will contain the number of simple string variables, the string array ST their names. These were initialized in 2.15.
2.48	For string arrays, 2.54 is first executed.
2.54	N will contain the total number of arrays (string , integer or floating), V their names.
2.48	I3 is an array that will contain zero for string arrays, 7 for integer arrays, 13 for floating arrays. Note that 7 and 13 represent the width foreseen in the output listing for integers and floating point numbers.
2.50-2.65	See above. For integer and floating arrays, the programme continues with 2.58.
2.58	IM will contain the maximum number of columns that may be printed side by side (for 2-dimensional arrays). LM is reduced by 8 to make space for the column containing the element numbers and for the inter-column spacing.
2.60	T is further decoded. D1 and D2 will contain the array dimensions (as strings). D2 will be 1 for one-dimensional arrays.
2.62	A will contain the first dimension (i.e. the number of lines needed in the listing), I4 the second one. If the latter is so high that not all columns may be printed side by side in a single matrix, the array is treated (and later listed) as several different sub-arrays, all with the same name, and each containing a number of columns that can fit inside a page width. I1 will contain the index

corresponding to the first column, I2 the one corresponding to the last column of the sub-array.

2.64 If the whole array will fit into a single matrix, I2 is simply equal to I4.

2.68 S will contain the lateral space needed for printing the array (or sub-array), excluding the first column with element numbers.

2.66 If I4 was too large for a single matrix, I2 is now defined and 2.68 will again define the total width S.

2.70 The number of arrays is then increased by one. The variable name V and the "vertical" size A are copied for the next sub-array.

2.72 I1 and I3 are also defined and control goes to 2.64, where the same process repeats until the entire original array is treated.

3.05 Reached from 2.2 when the analysis of the LISV output is completed. For each simple string variable, group 5 is executed.

5.10 First, the variable name (saved in array ST) is typed. The contents of the string are placed into E.

5.20 Group 29 will decode the string, differentiating between control characters and normal string contents. This group is also used for string arrays (called from 11.1).

29.10-29.20 In principle, an entire 80-character string might have to be printed in the form of 4-character control sequences like, e.g. \132. The decoded string (in the form in which it will be listed) would then contain 320 characters.
To provide intermediate storage for this, a string array C (defined in 5.2) is used. Note that for string arrays, all elements must be decoded before they may be printed, because the width reserved for an array table depends on its largest decoded element. Each of these decoded strings will be temporarily stored in C, starting at a position C(SH). For array elements, C(SH), C(SH+1), C(SH+2), C(SH+3) and C(SH+4) are reserved for the decoded string; C(SH+5) will contain the corresponding element number. In the present case of a simple string variable, SH = 1 (see 5.2).

29.10 Decoding a string is a complicated process that may take a considerable time. Group 29 therefore first tries if a string contains no "control sequences" in which case the decoding is simple and quick. Y is predefined and saved with the program; it contains all characters up to \174, except those in the range from \32 to \126, which are not considered as control characters.

29.20 If the match is unsuccessful (i.e. no control characters), this line provides the decoding. It just adds the quotes to the original string E. Q will contain the size of the decoded string.

Note that E might contain 79 or 80 characters. In that case, an error would occur, and group 30 would recover (see 5.2). Group 30 is also called from 29.1 if control characters are found.

30.10 Initialization. CF contains the string size, NC points to the first element of C to be used. This is emptied, and the flag FL (used for treating the \9 and \11 sequences) is set to zero.

30.20 Q will contain the size of the decoded string.

30.30 All characters are looked at by group 31.

31.10 AS will contain the ASCII equivalent. The next three lines are skipped if FL is equal to 1, i.e. if the preceding character was \9 or \11.

31.20 If the character is a "printable" one, AS is made negative, and ROF will return control to 30.4.

31.30-31.40 Otherwise, a check on \9 and \11 is made first.

31.50 If sufficient space remains in C(NC), line 31-6 does the decoding by adding the contents of Q1(AS) to C(NC). Q1 is predefined; it contains the decoded control characters.

30.72 If C(NC) did not contain at least 4 unused spaces, this line will increase NC by one and empty the corresponding C element after updating Q.

30.40 If a non-control character was found, AS is negative and control goes to 30.5. If AS is zero or positive, this means that the FOR command of 30.3 has been terminated in the normal way, so that all characters have been treated (the last one being a control character). Then, 30.9 will end the process by updating the total size Q.

30.50 If character CH was of non-control type, this line will put all successive non-control characters into ZZ, starting from CH and up to the next control character.

30.60 ZZ will contain nothing if no more control characters are found. This case is catered for here.

30.70 Otherwise, if space remains, a quote is first added to C(NC).

30.72-30.74 If not, the next C element is taken as before, and the process repeats.

30.76-30.78 Similarly, ZZ is added to C.

30.80-30.82 The same for the second quote. Note that the three operations 30.7 - 30.82 cannot be combined, because ZZ might contain more than 78 characters.

30.84 CH is updated and the process (unless finished) is continued with line 30.3.

5.30 LI is the maximum number of characters that may be printed on a line (8 being needed for the variable name and a small space). Group 6 will list each relevant C element; line 6.5 recovers if an undefined one is encountered (i.e. if the string needed less than 5 elements).

6.10 The C element is put into E. SP is only used for string arrays, not for simple string variables. (It was initialized in 2.15, however, to prevent errors here).

6.20 If E fits on a single line, it is typed after the variable name (which was already typed by 5.1). Its size is then subtracted from LI (only used for string arrays).

6.30 If the string was too long, its first part is printed, some space is made on the next line (below the variable name or, for arrays, the element number), and the rest of the string is put into E again.

6.40 The available space is redefined and the process continues. An illustration of the result may be seen at variable Y.987 in the listing of LISTER itself.

3.07 If no arrays are to be listed, control goes to 3.7.

3.10 Initialization. F(N) will contain the first element number of C used for string array no. N. Array C is emptied again because it might still contain something from the simple string variables.

3.15 SH is also initialized. For each string array, group 10 is done.

10.10 This group does the decoding. It also finds which elements are defined; only these will be listed. A contains the number of defined elements (and therefore the number of lines in the table, cf. 2.62).

10.20-10.30 Each element (J=0, J=1, ...) of the array will be considered. If it is defined, MF will be increased by one, and the element will be decoded. The largest decoded size found will be kept in S. (Minimum 6, to obtain a correct layout, even if not a single element is defined). The present C pointer is saved in array F for later use (in 4.3). Group 11 is done for each element, until all defined ones have been found. Recovery for errors in group 11 (undefined elements) is provided.

11.10 The element is transferred to E and the decoding is done by groups 29 and 30 as described before. The maximum size S, and the number of defined elements found MF are updated. The element number and a double space are kept in C(SH+5) and, finally, SH is increased by 6 for use by the next element.

3.20 All string arrays have been decoded, so that their table width S is now known. The tables will be printed in batches, each batch containing a group of arrays that will fit side by side across the width of the page. P will contain the serial number of the first array to be output in the next batch, Q the one for the last array. Q is first set to zero, and lines 3.25 - 3.65 will be executed repeatedly, once for each batch.

3.25 P is defined. L will contain the total width needed for all arrays of this batch considered so far.

3.30	The next array is taken. If no more exist, control goes to 3.4 and printing starts.
3.35	The width is increased by S and by 12, providing room for the column with the element numbers, and for 4 blanks (inter-table spacing). If L becomes too large for the page width, control goes to 3.4. Note that LM+4 is used, for the comparison, because the inter-table spacing is not needed for the last table of the batch.
3.40	Since the last array considered was too much for the width available, Q must be decreased by 1. However, if the batch only contains a single array, this is of course not done. The width needed for a single array may be larger than LM, if it is a string array whose decoded elements may contain up to 320 characters. Such long strings will be spread over several lines.
3.45	The array names are typed above the tables by group 12.
12.10	The name is printed. If it is the last one of the batch, this is all.
12.20	Otherwise, the name is followed by a space of SP blanks, calculated so that the next name will be printed in the correct place. Group 15 prints this space (which may contain more than 80 blanks and may therefore not be produced by a simple T&SP command).
3.47	After a line feed, group 14 will print the second subscripts of two-dimensional arrays above their corresponding columns.
14.10	For string arrays, only a space will be produced by 14.5.
14.20	The same for one-dimensional arrays.
14.25	A different space is made for integer and floating arrays, to obtain a correct layout.
14.30	The second subscripts are printed.
14.35	Additional space for floating arrays.
14.60	Inter-table space (not for last array of batch).
3.50	J was made zero in 3.47. If any second subscripts were printed, it is now larger than zero (see 14.3) and an additional line feed is made. R will be made equal to the largest table size A of the present batch; it is made zero first.
3.55	Finding the largest table size.
3.60	For each line I, group 4 prints the elements; J denotes the serial number of the array.
4.10	If I is larger than A(J), there are no more elements for this array, and a space consisting of S(J)+8 blanks must be printed.
4.50	This is not necessary for the last array of the batch.
4.60	As explained before, four blanks for inter-table spacing.
4.20	For numerical arrays, the next lines are skipped.
4.30	SH is the pointer for C at which this decoded element is stored. SP is the maximum width for the array table concerned.

4.40	The element number is typed and the available page width LI is defined. Then, as explained before, group 6 prints the string, using more than one line if necessary. The remaining space will then be in SP (see line 6.1); lines 4.5 and 4.6 as before.
4.70	This line is reached from 4.2 if the array is a numerical one. The element number (or first subscript) is typed and for each value of the second subscript group 8 is done.
8.10	$V(J) is the array name; B will contain the value to be printed.
8.20	For integer arrays.
8.30-8.40	If the array is a floating one, a check is made for very small or very large numbers; these are printed in a somewhat reduced E format by line 8.6.
8.50	Otherwise, a fixed-point format is chosen.
4.80	After each array, a space of four blanks.
3.65	If any arrays remain, continue with the next batch.
3.70	If there are no defined functions, restart for the next file. The RUN command erases all information from the file that was just listed to avoid clashes.
3.80	If there are defined functions to be listed, the file is loaded into the defined function area.
3.90	Then, T1 (predefined and saved with LISTER) is executed and the program restarted as before.
T1	For each defined function the relevant LISV line is printed (defined in 2.42). The function is then opened and listed. Since the OPEN command erases the LISTER programme, the whole T1 sequence must be executed by a $DO. The two nested $DO's are needed because the RUN command in 3.9 cannot follow at the end of T1; it should only be executed after the entire FOR loop contained in T1 is satisfied. The first $DO ensures that the RUN command is not erased by the OPEN command.