Fortran Character String Utilities

A collection of string manipulation routines is contained in the module 'strings' found in the file **stringmod.f90**. To obtain this module as well as some other string utilities, go to the website http://www.gbenthien.net/strings/index.html. To use the routines in the module 'strings' the user needs to add the statement

Use strings

to the top of the program. These routines were developed primarily to aid in the reading and manipulation of input data from an ASCII text file. The routines are described below.

1. SUBROUTINE PARSE(*str, delims, args, nargs*)

   This routine was originally designed to separate the arguments in a command line where the arguments are separated by certain delimiters (commas, spaces, etc.). However, this routine can be used to separate other types of strings into their component parts. The first input is a string *str* (e.g., a command line). The second argument is a string *delims* containing the allowed delimiters. For example, *delims* might be the string ′ ,′ consisting of a comma and a space. The third argument is a character array *args* that contains on output the substrings (arguments) separated by the delimiters. Initial spaces in the substrings (arguments) are deleted. The final argument is an integer *nargs* that gives the number of separated parts (arguments). To treat a delimiter in *str* as an ordinary character precede it by a backslash (\). If a backslash character is desired in *str*, precede it by another backslash (\\). In addition, spaces that immediately precede or follow another delimiter are not considered delimiters. Multiple spaces or tabs are considered as a single space, i.e., ′a b′ is treated the same as ′a b′. Backslashes can be removed from an argument by calling the routine REMOVEBKSL, i.e.,

   call removebksl(<string>)

   This routine converts double backslashes (\\) to single backslashes (\).

   Example: If the delimiters are a comma and a space (*delims* = ′ ,′), then the subroutine parse applied to the string ′cmd arg1, arg\ 2, arg3′ produces the output

   args(1) = ′cmd′
   args(2) = ′arg1′
   args(3) = ′arg 2′
   args(4) = ′arg3′
   nargs = 4

2. SUBROUTINE COMPACT(*str*)

   This routine converts multiple spaces and tabs to single spaces and deletes control characters.

3. SUBROUTINE REMOVESP(*str*)

   This routine removes spaces, tabs, and control characters in string *str*.

4. SUBROUTINE VALUE(*str, number, ios*)

   This subroutine converts a number string to a number. The argument *str* is a string representing a number. The argument *number* is the resulting real number or integer (single or double precision). The argument *ios* is an error flag. If *ios* is nonzero, then there was an error in the conversion.

5. SUBROUTINE SHIFTSTR(*str, n*)

   This routine shifts characters in the string *str* by *n* positions (positive values denote a right shift and negative values denote a left shift). Characters that are shifted off the end are lost. Positions opened up by the shift are replaced by spaces.

6. SUBROUTINE INSERTSTR(*str, strins, loc*)

   This routine inserts the string *strins* into the string *str* at position *loc*. Characters in *str* starting at position *loc* are shifted right to make room for the inserted string.

7. SUBROUTINE DELSUBSTR(*str, substr*)

   This subroutine deletes the first occurrence of substring *substr* from string *str* and shifts characters left to fill hole.

8. SUBROUTINE DELALL(*str*, *substr*)

   This routine deletes all occurrences of substring *substr* from string *str* and shifts characters left to fill holes.

9. FUNCTION UPPERCASE(*str*)

   This function returns a string that is like the string *str* with all characters that are not between a pair of quotes ("…" or '…') converted to uppercase.

10. FUNCTION LOWERCASE(*str*)

    This function returns a string that is like the string *str* with all characters that are not between a pair of quotes ("…" or '…') converted to lowercase.

11. SUBROUTINE READLINE(*nunitr, line, ios*)

    This routine reads a line from unit *nunitr*, ignoring blank lines and deleting comments beginning with an exclamation point(!). The line is placed in the string *line*. The argument *ios* is an error flag. If *ios* is not equal to zero, then there has been an error in the read operation. A negative value for *ios* denotes an end of file.

12. SUBROUTINE MATCH(*str, ipos, imatch*)

    This routine finds the delimiter in string *str* that matches the delimiter in position *ipos* of *str*. The argument *imatch* contains the position of the matching delimiter. Allowable delimiters are (), [], {}, <>.

13. SUBROUTINE WRITENUM(*number, string, fmt*)

    This routine writes a number to a string. The argument *number* is a real number or an integer (single or double precision). The number *number* is written to the character string *string* with format *fmt* (e.g., 'e15.6' or 'i5').

14. SUBROUTINE TRIMZERO(*str*)

    This subroutine deletes nonsignificant trailing zeroes in a number string *str*. A single zero following a decimal point is allowed. For example, ′1.50000′ is converted to ′1.5′ and ′5.′ is converted to ′5.0′.

15. SUBROUTINE WRITEQ(*unit, name, value, fmt*)

This routine writes a string of the form '<name> = <value>' to the unit *unit*. Here <name> is the input string *name* and value is the input number *value* converted to a string with the format *fmt*. The number value can be a real number or an integer (single or double precision).

16. FUNCTION IS_LETTER(*ch*)

This function returns the logical value .true. if the input character *ch* is a letter (a–z or A–Z). It returns the value .false. otherwise.

17. FUNCTION IS_DIGIT(*ch*)

This function returns the logical value .true. if the input character *ch* is a digit (0–9). It returns the value .false. otherwise.

18. SUBROUTINE SPLIT(*str, delims, before, sep*)

This routine uses the first occurrence of a character from the string *delims* in the string *str* to split the string into two parts. The portion of *str* before the found delimiter is output in *before;* the portion of *str* after the found delimiter is output in *str* (*str* is left justified). The output character *sep* (optional) contains the found delimiter. To treat a delimiter in *str* as an ordinary character precede it by a backslash (\). If a backslash is desired in *str*, precede it by another backslash (\\). Repeated applications of SPLIT can be used to parse a string into its component parts. Backslashes can be removed by calling the routine REMOVEBKSL, i.e.,

call removebksl(<string>)