

International Standards Organization

Varying Length Character Strings

in

Fortran

ISO/IEC 1539-2

{auxiliary standard to ISO/IEC 1539 : 1991}

{Second Committee Draft Produced 26-Jul-93}

Contents

Foreword	v
Introduction	vi
Section 1 : Scope	1
1.1 Normative References	2
Section 2 : Requirements	3
2.1 The Name of the Module	3
2.2 The Type	3
2.3 Extended Meanings for Intrinsic Operators	3
2.3.1 Assignment	3
2.3.2 Concatenation	4
2.3.3 Comparisons	4
2.4 Extended Meanings for Generic Intrinsic Procedures	4
2.4.1 The LEN Procedure	4
2.4.2 The CHAR Procedure	5
2.4.3 The ICHAR Procedure	5
2.4.4 The IACHAR Procedure	5
2.4.5 The TRIM procedure	5
2.4.6 The LEN_TRIM procedure	6
2.4.7 The ADJUSTL procedure	6
2.4.8 The ADJUSTR procedure	6
2.4.9 The REPEAT procedure	7
2.4.10 Comparison Procedures	7
2.4.11 The INDEX procedure	8
2.4.12 The SCAN procedure	8
2.4.13 The VERIFY procedure	9
2.5 Additional Generic Procedure for Type Conversion	9
2.5.1 The VAR_STR procedure	9
2.6 Additional Generic Procedures for Input/Output	9
2.6.1 The GET procedure	10
2.6.2 The PUT procedure	11
2.6.3 The PUT_LINE procedure	11
2.7 Additional Generic Procedures for Substring Manipulation	12
2.7.1 The INSERT procedure	12
2.7.2 The REPLACE procedure	12
2.7.3 The REMOVE procedure	13
2.7.4 The EXTRACT procedure	14
2.7.5 The SPLIT procedure	14
Annex A	15
MODULE ISO_VARYING_STRING	15

Annex B	65
PROGRAM word_count	65
PROGRAM vocabulary_word_count	65

Foreword

[This page to be provided by ISO CS]

Introduction

1

2 This International Standard has been prepared by ISO/IEC JTC1/SC22/WG5, the technical working
3 group for the Fortran language. This International Standard is an auxiliary standard to
4 ISO/IEC 1539 : 1991, which defines the latest revision of the Fortran language, and is the first part
5 of the multipart Fortran family of standards; this International Standard is the second part. The revised
6 language defined by the above standard is informally known as Fortran 90.

7 This International Standard defines the interface and semantics for a module which provides facilities
8 for the manipulation of character strings of arbitrary and dynamically variable length. The annex A
9 includes a possible implementation in Fortran 90 of a module that conforms to this International
10 Standard. It should be noted, however, that this is purely for purposes of demonstrating the feasibility
11 and portability of the standard. The actual code shown in this annex is not intended in any way to
12 prescribe the method of implementation, nor is there any implication that this is in any way an optimal
13 portable implementation. The module is merely a fairly straight forward demonstration that a portable
14 implementation is possible.

Section 1 : Scope

This International Standard defines facilities for use in Fortran for the manipulation of character strings of dynamically variable length. This International Standard provides an auxiliary standard for the Fortran language informally known as Fortran 90. The standard defining this revision of the Fortran language is

- ISO/IEC 1539 : 1991 "Programming Language Fortran"

This International Standard is an auxiliary standard to that defining Fortran 90 in that it defines additional facilities to those defined intrinsically in the primary language standard. However, a processor conforming to the Fortran 90 standard is not required to also conform to this International Standard. Nevertheless, conformance to this International Standard assumes conformance to the primary Fortran 90 standard.

This International Standard prescribes the name of a Fortran module, the name of the derived data type to be used to represent varying-length strings, the interfaces for the procedures and operators that must be provided to manipulate objects of this type, and the semantics that are required for each of the entities made accessible by this module.

This International Standard does not prescribe the details of any implementation. Neither the method used to represent the data entities of the defined type nor the algorithms used to implement the procedures or operators whose interfaces are defined by this International Standard are prescribed. A conformant implementation may use any representation and any algorithms, subject only to the requirement that the publicly accessible names and interfaces conform to this International Standard, and that the semantics are as required by this International Standard and those of ISO/IEC 1539 : 1991.

It should be noted that a processor is not required to implement this International Standard in order to be a standard conforming Fortran processor, but if a processor implements facilities for manipulating varying length character strings, it is recommended that this be done in a manner that is conformant with this International Standard. A processor conforming to this International Standard may extend the facilities provided for the manipulation of varying length character strings as long as such extensions do not conflict with those defined in this International Standard.

A module, written in standard conforming Fortran, is included in Annex A. This module illustrates one way in which a standard conforming module could be written. This module is both conformant with the requirements of this International Standard and, because it is written in standard conforming Fortran, it provides a portable implementation of the required facilities.

This module is included for information only and is not intended to constrain implementations in any way. This module is a demonstration that at least one implementation, in standard conforming and hence portable Fortran, is possible.

It should be noted that this International Standard defines facilities for dynamically varying length strings of characters of default kind only. Throughout this International Standard all references to intrinsic type **CHARACTER** should be read as meaning characters of default kind. Similar facilities could be defined for non-default kind characters by a separate, if similar, module for each such character kind.

1 This International Standard has been designed, as far as is reasonable, to provide for varying length
2 character strings the facilities that are available for intrinsic fixed length character strings. All the
3 intrinsic operations and functions which apply to fixed length character strings have extended meanings
4 defined by this International Standard for varying length character strings. Also a small number of
5 additional facilities are defined that are appropriate because of the essential differences between the
6 intrinsic type and the varying length derived data type.

7 **1.1 Normative References**

- 8 - ISO/IEC 1539 : 1991 "Programming Language Fortran"
- 9 - ISO/IEC 646 : 1983 "Character Coding"

Section 2 : Requirements

2.1 The Name of the Module

The name of the module shall be

```
ISO_VARYING_STRING
```

Programs shall be able to access the facilities defined by this International Standard by the inclusion of **USE** statements of the form

```
USE ISO_VARYING_STRING
```

2.2 The Type

The type shall have the name

```
VARYING_STRING
```

Entities of this type shall represent values which are strings of characters of default kind. These character strings may be of any non-negative length and this length may vary dynamically during the execution of a program. There shall be no arbitrary upper length limit other than that imposed by the size of the processor and the complexity of the programs it is able to process. The characters representing the value of the string have positions 1,2,...,N, where N is the length of the string. The internal structure of the type shall be **PRIVATE** to the module.

2.3 Extended Meanings for Intrinsic Operators

The meanings for the intrinsic operators of:

```
assignment      =
concatenation   //
comparisons     ==, /=, <, <=, >=, >
```

shall be extended to accept any combination of scalar operands of type **VARYING_STRING** and type **CHARACTER**. Note that, the equivalent comparison operator forms, **.EQ.**, **.NE.**, **.LT.**, **.LE.**, **.GE.**, **.GT.**, also have their meanings extended in this manner.

2.3.1 Assignment: An assignment of the form

```
var = expr
```

shall be defined for scalars with the following type combinations:

```
VARYING_STRING = VARYING_STRING
VARYING_STRING = CHARACTER
CHARACTER = VARYING_STRING
```

Action: The characters that are the value of the expression **expr** become the value of the variable **var**. There are two cases:

- Case(i) : Where the variable is of type **VARYING_STRING**, the length of the variable becomes that of the expression.
- Case(ii) : Where the variable is of type **CHARACTER**, the rules of intrinsic assignment to a Fortran character variable apply. Namely, if the expression string is longer than the declared length of the character variable, only the left-most characters are assigned. If the character variable is longer than that of the string expression, it is padded on the right with blanks.

2.3.2 Concatenation: The concatenation operation`string_a // string_b`

shall be defined for scalars with the following type combinations:

`VARYING_STRING // VARYING_STRING``VARYING_STRING // CHARACTER``CHARACTER // VARYING_STRING`

The values of the operands are unchanged by the operation.

Result Type: `VARYING_STRING`**Result Value:** The result value is a new string whose characters are the same as those produced by concatenating the operand character strings in the order given.**2.3.3 Comparisons:** Comparisons of the form`string_a .OP. string_b`where `.OP.` represents any of the operators `==`, `/=`, `<`, `<=`, `>=`, `>` shall be defined for scalar operands with the following type combinations:`VARYING_STRING .OP. VARYING_STRING``VARYING_STRING .OP. CHARACTER``CHARACTER .OP. VARYING_STRING`

The values of the operands are unchanged by the operation.

Note that, the equivalent operator forms `.EQ.`, `.NE.`, `.LT.`, `.LE.`, `.GE.`, `.GT.` also have their meanings extended in this manner.**Result Type:** default `LOGICAL`.**Result Value:** The result value is true if `string_a` stands in the indicated relation to `string_b`. The collating sequence used for the inequality comparisons is that defined by the processor for characters of default kind. If `string_a` and `string_b` are of different lengths, the comparison is done as if the shorter string were padded on the right with blanks.**2.4 Extended Meanings for Generic Intrinsic Procedures**The generic intrinsic procedures `LEN`, `CHAR`, `ICHAR`, `IACHAR`, `TRIM`, `LEN_TRIM`, `ADJUSTL`, `ADJUSTR`, `REPEAT`, `LLT`, `LLE`, `LGE`, `LGT`, `INDEX`, `SCAN`, and `VERIFY` shall have their meanings extended to include the appropriate scalar argument type combinations involving `VARYING_STRING` and `CHARACTER`.**2.4.1 The LEN Procedure:** The generic function reference of the form`LEN(string)`

shall be added.

Description: Returns the length of a character string.**Class:** Transformational function.**Argument:** `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by the procedure.**Result Type:** default `INTEGER`.**Result Value:** The result value is the number of characters in `string`.

2.4.2 The CHAR Procedure: The generic function references of the form

`CHAR(string)`
`CHAR(string,length)`

shall be added.

Description: Converts a varying string value to default character.

Class: Transformational function.

Arguments:

`string` - is of type `VARYING_STRING`

`length` - is of type default `INTEGER`.

The arguments must be scalars and are unchanged by the procedure.

Result Type: default `CHARACTER`.

Result Value:

Case(i) : If `length` is absent, the result has the value of the characters of `string`, and the same length.

Case(ii) : If `length` is present, the result has the length specified by the argument `length`. If `string` is longer than `length`, the result is truncated on the right. If `string` is shorter than `length`, the result is padded on the right with blanks. If `length` is less than one, the result is of zero length.

2.4.3 The ICHAR Procedure: The generic function reference of the form

`ICHAR(c)`

shall be added.

Description: Position of a character in the processor collating sequence.

Class: Transformational function.

Argument: `c` is a scalar of type `VARYING_STRING` and of length exactly one. The argument is unchanged by the procedure.

Result Type: default `INTEGER`.

Result Value: The result value is the position of the character `c` in the processor defined collating sequence for default characters.

2.4.4 The IACHAR Procedure: The generic function reference of the form

`IACHAR(c)`

shall be added.

Description: Position of a character in the collating sequence defined by the standard ISO 646 : 1983.

Class: Transformational function.

Argument: `c` is a scalar of type `VARYING_STRING` and of length exactly one. The argument is unchanged by the procedure.

Result Type: default `INTEGER`.

Result Value: The result value is the position of the character `c` in the collating sequence defined by the standard ISO 646 : 1983 for default characters. If the character `c` is not defined in the standard set, the result is processor dependent.

2.4.5 The TRIM procedure: The generic function reference of the form

`TRIM(string)`

shall be added.

Description: Remove trailing blanks from a string.

Class: Transformational Function.

Argument: `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by the procedure.

1 **Result Type:** `VARYING_STRING`.

2 **Result Value:** The result value is the string produced by removing any trailing blanks
3 from the argument. If the argument `string` contains only blank characters or is of zero
4 length, the result is a zero-length string.

5 **2.4.6 The `LEN_TRIM` procedure:** The generic function reference of the form

6 `LEN_TRIM(string)`
7 shall be added.

8 **Description:** Length of a string not counting any trailing blanks.

9 **Class:** Transformational function.

10 **Argument:** `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by
11 the procedure.

12 **Result Type:** default `INTEGER`.

13 **Result Value:** The result value is the position of the last non-blank character in `string`.
14 If the argument `string` contains only blank characters or is of zero length, the result is
15 zero.

16 **2.4.7 The `ADJUSTL` procedure:** The generic function reference of the form

17 `ADJUSTL(string)`
18 shall be added.

19 **Description:** Adjusts to the left, removing any leading blanks and inserting trailing blanks.

20 **Class:** Transformational function.

21 **Argument:** `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by
22 the procedure.

23 **Result Type:** `VARYING_STRING`.

24 **Result Value:** The result value contains the same characters as the argument shifted
25 cyclically to the left until the first character is non-blank. The result is identical to the
26 argument if the first character of `string` is non-blank, `string` contains only blank
27 characters or is of zero length.

28 **2.4.8 The `ADJUSTR` procedure:** The generic function reference of the form

29 `ADJUSTR(string)`
30 shall be added.

31 **Description:** Adjusts to the right, removing any trailing blanks and inserting leading
32 blanks.

33 **Class:** Transformational function.

34 **Argument:** `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by
35 the procedure.

36 **Result Type:** `VARYING_STRING`.

37 **Result Value:** The result value contains the same characters as the argument shifted
38 cyclically to the right until the last character is non-blank. The result is identical to the
39 argument if the last character of `string` is non-blank, `string` contains only blank
40 characters or is of zero length.

1 **2.4.9 The REPEAT procedure:** The generic function reference of the form

2 **REPEAT**(*string*,*ncopies*)

3 shall be added.

4 **Description:** Concatenate several copies of a string.

5 **Class:** Transformational function.

6 **Arguments:**

7 *string* - is a scalar of type **VARYING_STRING**,

8 *ncopies* - is a scalar of type default **INTEGER**.

9 The value of *ncopies* must not be negative. The arguments are unchanged by the
10 procedure.

11 **Result Type:** **VARYING_STRING**.

12 **Result Value:** The result value is the string produced by repeated concatenation of the
13 argument *string*, producing a string containing *ncopies* copies of *string*. A negative
14 value for *ncopies* is not permitted. If *ncopies* is zero, the result is of zero length.

15 **2.4.10 Comparison Procedures:** The set of generic function references of the form

16 **Lop**(*string_a*,*string_b*)

17 shall be added, where *op* stands for one of:

18 **LT** - less than

19 **LE** - less than or equal to

20 **GE** - greater than or equal to

21 **GT** - greater than

22 **Description:** Compares the lexical ordering of two strings based on the ISO 646 : 1983
23 collating sequence.

24 **Class:** Transformational function.

25 **Arguments:** *string_a* and *string_b* are scalars of one of the type combinations:

26 **VARYING_STRING** and **VARYING_STRING**,

27 **VARYING_STRING** and **CHARACTER**, or

28 **CHARACTER** and **VARYING_STRING**.

29 The arguments are unchanged by the procedure.

30 **Result Type:** default **LOGICAL**.

31 **Result Value:** The result value is true if *string_a* stands in the indicated relationship to
32 *string_b*, and is false otherwise. The collating sequence used to establish the ordering
33 of characters for these procedures is that of the International Standard, ISO 646 : 1983.
34 If *string_a* and *string_b* are of different length, the comparison is done as if the shorter
35 string were padded on the right with blanks. If either argument contains a character not
36 defined by the standard, the result value is processor dependent.

1 **2.4.11 The INDEX procedure:** The generic function reference of the form

2 INDEX(*string*,*substring*,*back*)

3 shall be added.

4 **Description:** Returns an integer which is the starting position of a substring within a
5 string.

6 **Class:** Transformational function.

7 **Arguments.** *string* and *substring* are scalars of one of the type combinations:

8 VARYING_STRING and VARYING_STRING,
9 CHARACTER and VARYING_STRING, OR
10 VARYING_STRING and CHARACTER.

11 *back* - is a scalar of type default LOGICAL and is OPTIONAL.

12 **Result type:** default INTEGER.

13 **Result value:**

- 14 Case(i) : If *back* is absent or is present with the value false, the result is the
15 minimum positive value of *i* such that,
16 EXTRACT(*string*,*i*,*i*+LEN(*substring*)-1)==*substring*,
17 or zero if there is no such value. Zero is returned if
18 LEN(*string*)<LEN(*substring*), and one is returned if
19 LEN(*substring*)=0.
- 20 Case(ii) : If *back* is present with the value true, the result is the maximum value
21 of *i* less than or equal to LEN(*string*)-LEN(*substring*)+1 such that
22 EXTRACT(*string*,*i*:*i*+LEN(*substring*)-1)==*substring*,
23 or zero if there is no such value. Zero is returned if
24 LEN(*string*)<LEN(*substring*), and LEN(*string*)+1 is returned if
25 LEN(*substring*)=0.

26 **2.4.12 The SCAN procedure:** The generic function reference of the form

27 SCAN(*string*,*set*,*back*)

28 shall be added.

29 **Description:** Scan a string for any one of the characters in a set of characters.

30 **Class:** Transformational function.

31 **Arguments:** *string* and *set* are scalars of one of the type combinations:

32 VARYING_STRING and VARYING_STRING,
33 VARYING_STRING and CHARACTER, OR
34 CHARACTER and VARYING_STRING.

35 *back* - is a scalar of type default LOGICAL and is OPTIONAL.

36 The arguments are unchanged by the procedure.

37 **Result Type:** default INTEGER.

38 **Result Value:**

- 39 Case(i) : If *back* is absent or is present with the value false and if *string*
40 contains at least one character that is in *set*, the value of the result is
41 the position of the leftmost character of *string* that is in *set*.
- 42 Case(ii) : If *back* is present with the value true and if *string* contains at least
43 one character that is in *set*, the value of the result is the position of
44 the rightmost character of *string* that is in *set*.
- 45 Case(iii) : The value of the result is zero if no character of *string* is in *set* or
46 if the length of either *string* or *set* is zero.

1 **2.4.13 The VERIFY procedure:** The generic function reference of the form

2 **VERIFY**(*string*,*set*,*back*)

3 shall be added.

4 **Description:** Verify that a string contains only characters from a given set by scanning
5 for any character not in the set.

6 **Class:** Transformational function.

7 **Arguments:** *string* and *set* are scalars of one of the type combinations:

8 **VARYING_STRING** and **VARYING_STRING**,

9 **VARYING_STRING** and **CHARACTER**, or

10 **CHARACTER** and **VARYING_STRING**.

11 *back* - is a scalar of type default **LOGICAL** and is **OPTIONAL**.

12 The arguments are unchanged by the procedure.

13 **Result Type:** default **INTEGER**.

14 **Result Value:**

15 Case(i) : If *back* is absent or is present with the value false and if *string*
16 contains at least one character that is not in *set*, the value of the result
17 is the position of the leftmost character of *string* that is not in *set*.

18 Case(ii) : If *back* is present with the value true and if *string* contains at least
19 one character that is not in *set*, the value of the result is the position
20 of the rightmost character of *string* that is not in *set*.

21 Case(iii) : The value of the result is zero if each character of *string* is in *set* or
22 if the length of *string* is zero.

23 **2.5 Additional Generic Procedure for Type Conversion**

24 An additional generic procedure shall be added to convert scalar intrinsic fixed-length character values
25 into scalar varying-length string values.

26 **2.5.1 The VAR_STR procedure:** The generic function reference of the form

27 **VAR_STR**(*char*)

28 shall be provided.

29 **Description:** Converts an intrinsic fixed-length character value into the equivalent
30 varying-length string value.

31 **Class:** Transformational function.

32 **Argument:** *char* is a scalar of type default **CHARACTER** and may be of any length. The
33 argument is unchanged by the procedure.

34 **Result Type:** **VARYING_STRING**.

35 **Result Value:** The result value is the same string of characters as the argument.

36 **2.6 Additional Generic Procedures for Input/Output**

37 The following additional generic procedures shall be provided to support input and output of
38 varying-length string values with formatted sequential files.

39 **GET** - input part or all of a record into a string

40 **PUT** - append a string to an output record

41 **PUT_LINE** - append a string to an output record and end the record

1 **2.6.1 The GET procedure:** The generic subroutine references of the forms

2 CALL GET(*string*,*maxlen*,*iostat*)
 3 CALL GET(*unit*,*string*,*maxlen*,*iostat*)
 4 CALL GET(*string*,*set*,*maxlen*,*iostat*)
 5 CALL GET(*unit*,*string*,*set*,*maxlen*,*iostat*)

6 shall be provided.

7 **Description:** Input characters from an external file into a string.

8 **Class:** Subroutine.

9 **Arguments:**

10 *string* - is of type **VARYING_STRING**,
 11 *maxlen* - is of type default **INTEGER** and is **OPTIONAL**,
 12 *unit* - is of type default **INTEGER**,
 13 *set* - is either of type **VARYING_STRING** or of type **CHARACTER**
 14 *iostat* - is of type default **INTEGER** and is **OPTIONAL**.

15 All arguments are scalar. The argument *unit* specifies the input unit to be used. It must
 16 be connected to a formatted file for sequential read access. If the argument *unit* is
 17 omitted, the default input unit is used.

18 **Action:** The **GET** procedure causes characters from the connected file, starting with the
 19 next character in the current record if there is a current record or the first character of the
 20 next record if not, to be read and stored in the variable *string*. The end of record always
 21 terminates the input but input may be terminated before this. If *maxlen* is present, its
 22 value indicates the maximum number of characters that will be read. If *maxlen* is less than
 23 or equal to zero, no characters will be read and *string* will be set to zero length. If
 24 *maxlen* is absent, a maximum of **HUGE(1)** is used. If the argument *set* is provided, this
 25 specifies a set of characters the occurrence of any of which will terminate the input. This
 26 terminal character, although read from the input file, will not be included in the result
 27 string. The file position after the data transfer is complete is after the last character that
 28 was read. If the transfer was terminated by the end of record being reached, the file is
 29 positioned after the record just read. If present, the argument *iostat* is used to return the
 30 status resulting from the data transfer. A zero value is returned if a valid read operation
 31 occurs, a positive value if an error is caused, and a negative value if an end-of-file
 32 condition occurs. If *iostat* is absent and anything other than a valid read operation
 33 occurs, the program execution is terminated.

1 **2.6.2 The PUT procedure:** The generic subroutine references of the forms

2 CALL PUT(string,iostat)
3 CALL PUT(unit,string,iostat)

4 shall be provided.

5 **Description:** Output a string to an external file.

6 **Class:** Subroutine.

7 **Argument:**

8 string - is either type **VARYING_STRING** or type **CHARACTER**,

9 unit - is type default **INTEGER**,

10 iostat - is type default **INTEGER** and is **OPTIONAL**.

11 All arguments are scalar. The argument **unit** specifies the output unit to be used. It must
12 be connected to a formatted file for sequential write access. If the argument **unit** is
13 omitted, the default output unit is used.

14 **Action:** The **PUT** procedure causes the characters of **string** to be appended to the current
15 record, if there is a current record, or to the start of the next record if there is no current
16 record. The last character transferred becomes the last character of the current record,
17 which is the last record of the file. If present, the argument **iostat** is used to return the
18 status resulting from the data transfer. A zero value is returned if a valid write operation
19 occurs, and a positive value if an error occurs. If **iostat** is absent and anything other than
20 a valid write operation occurs, the program execution is terminated.

21 **2.6.3 The PUT_LINE procedure:** The generic subroutine references of the forms

22 CALL PUT_LINE(string,iostat)
23 CALL PUT_LINE(unit,string,iostat)

24 shall be provided.

25 **Description:** Output a string to an external file and end the record.

26 **Class:** Subroutine.

27 **Argument:**

28 string - is either type **VARYING_STRING** or type **CHARACTER**

29 unit - is type default **INTEGER**

30 iostat - is type default **INTEGER** and is **OPTIONAL**.

31 All arguments are scalar. The argument **unit** specifies the output unit to be used. It must
32 be connected to a formatted file for sequential write access. If the argument **unit** is
33 omitted, the default output unit is used.

34 **Action:** The **PUT_LINE** procedure causes the characters of **string** to be appended to the
35 current record, if there is a current record, or to the start of the next record if there is no
36 current record. Following completion of the data transfer, the file is positioned after the
37 record just written, which becomes the previous and last record of the file. If present, the
38 argument **iostat** is used to return the status resulting from the data transfer. A zero value
39 is returned if a valid write operation occurs, and a positive value if an error occurs. If
40 **iostat** is absent and anything other than a valid write operation occurs, the program
41 execution is terminated.

2.7 Additional Generic Procedures for Substring Manipulation

The following additional generic procedures shall be provided to support the manipulation of scalar substrings of scalar varying-length strings.

INSERT	-	insert a substring into a string
REPLACE	-	replace a substring in a string
REMOVE	-	remove a section of a string
EXTRACT	-	extract a section from a string
SPLIT	-	split a string into two at the occurrence of a separator

2.7.1 The INSERT procedure: The generic function reference of the form

INSERT(string,start,substring)

shall be provided.

Description: Insert a substring into a string at a specified position.

Class: Transformational function.

Arguments:

string - is either type **VARYING_STRING** or type default **CHARACTER**,

start - is type default **INTEGER**,

substring - is either type **VARYING_STRING** or type default **CHARACTER**.

All arguments must be scalars. The arguments are unchanged by the procedure.

Result Type: **VARYING_STRING**.

Result Value: The result value is a copy of the characters of the argument **string** with the characters of **substring** inserted into the copy of **string** before the character at the character position **start**. The remainder of the result string is shifted to the right and enlarged as necessary. If **start** is greater than **LEN(string)**, the value **LEN(string)+1** is used for **start** and the **substring** is appended to the copy of **string**. If **start** is less than one, the value one is used for **start** and the **substring** is inserted before the first character of the copy of the **string**. The length of the result is **LEN(string) + LEN(substring)**.

2.7.2 The REPLACE procedure: The generic function references of the forms

REPLACE(string,start,substring)

REPLACE(string,start,finish,substring)

REPLACE(string,target,substring,every,back)

shall be provided.

Description: Replaces a subset of the characters in a string by a given substring. The subset may be specified either by position or by content.

Class: Transformational function.

Arguments:

string - is either type **VARYING_STRING** or type default **CHARACTER**,

start - is type default **INTEGER**,

finish - is type default **INTEGER**,

substring - is either type **VARYING_STRING** or type default **CHARACTER**,

target - is either type **VARYING_STRING** or type default **CHARACTER**,

every - is type default **LOGICAL**, and is **OPTIONAL**,

back - is type default **LOGICAL**, and is **OPTIONAL**.

All arguments are scalar and are unchanged by the procedure. The argument **target** must not be of zero length. In all cases the arguments are unchanged by the procedure.

Result Type: **VARYING_STRING**.

1 **Result Value:** The result value is a copy of the characters in **string** modified as per one
2 of the cases below.

3 Case(i) : For a reference of the form

4 `REPLACE(string,start,substring)`

5 the characters of the argument **substring** are inserted in the copy of
6 **string** beginning with the character at the character position **start**.
7 The characters in positions from

8 `start to MIN(start+LEN(substring)-1,LEN(string))`

9 are deleted. The result string is enlarged if necessary. If **start** is
10 greater than `LEN(string)`, the value `LEN(string)+1` is used for **start**
11 and **substring** is appended to the copy of string. If **start** is less than
12 one, the value one is used for **start**.

13 Case(ii) : For a reference of the form

14 `REPLACE(string,start,finish,substring)`

15 the characters in the copy of **string** between positions **start** and
16 **finish**, including those at **start** and **finish**, are deleted and replaced
17 by the characters of **substring**. If **start** is less than one, the value
18 one is used for **start**. If **finish** is greater than `LEN(string)`, the
19 value `LEN(string)` is used for **finish**. If **finish** is less than **start**,
20 the characters of **substring** are inserted before the character at **start**
21 and no characters are deleted. The length of the result string is
22 adjusted as necessary.

23 Case(iii) : For a reference of the form

24 `REPLACE(string,target,substring,every,back)`

25 the copy of **string** is searched for occurrences of **target**. The search
26 is done in the backward direction if the argument **back** is present with
27 the value true, but in the forward direction otherwise. If **target** is
28 found, it is replaced by **substring**. If **every** is present with the value
29 true, the search and replace is continued from the character following
30 **target** in the search direction specified until all occurrences of **target**
31 in the copy string are replaced; otherwise only the first occurrence of
32 **target** is replaced.

33 2.7.3 The REMOVE procedure: The generic function reference of the form

34 `REMOVE(string,start,finish)`

35 shall be provided.

36 **Description:** Removes a specified substring from a string.

37 **Class:** Transformational function.

38 **Arguments:**

39 **string** - is either type `VARYING_STRING` or type default `CHARACTER`,

40 **start** - is type default `INTEGER`, and is `OPTIONAL`,

41 **finish** - is type default `INTEGER`, and is `OPTIONAL`.

42 All arguments must be scalars. The arguments are unchanged by the procedure.

43 **Result Type:** `VARYING_STRING`.

44 **Result Value:** The result value is a copy of the characters of **string** with the characters
45 between **start** and **finish**, inclusive, removed. If **start** is absent or less than one, the
46 value one is used for **start**. If **finish** is absent or greater than `LEN(string)`, the value
47 `LEN(string)` is used for **finish**. If **finish** is less than **start**, the characters of **string**
48 are delivered unchanged as the result.

1 **2.7.4 The EXTRACT procedure:** The generic function reference of the form

2 **EXTRACT**(string,start,finish)

3 shall be provided.

4 **Description:** Extracts a specified substring from a string.

5 **Class:** Transformational function.

6 **Arguments:**

7 string - is either type **VARYING_STRING** or type default **CHARACTER**,

8 start - is type default **INTEGER**, and is **OPTIONAL**,

9 finish - is type default **INTEGER**, and is **OPTIONAL**.

10 All arguments must be scalars. The arguments are unchanged by the procedure.

11 **Result Type:** **VARYING_STRING**.

12 **Result Value:** The result value is a copy of the characters of the argument **string**
13 between **start** and **finish**, inclusive. If **start** is absent or less than one, the value one
14 is used for **start**. If **finish** is absent or greater than **LEN(string)**, the value
15 **LEN(string)** is used for **finish**. If **finish** is less than **start**, the result is a zero-length
16 string.

17 **2.7.5 The SPLIT procedure:** The generic subroutine reference of the form

18 **CALL SPLIT**(string,word,set,separator,back)

19 shall be provided.

20 **Description:** Splits a string into a two substrings with the substrings separated by the
21 occurrence of a character from a specified separator set.

22 **Class:** Subroutine.

23 **Arguments:**

24 string - is type **VARYING_STRING**,

25 word - is type **VARYING_STRING**,

26 set - is either type **VARYING_STRING** or type default **CHARACTER**,

27 separator - is type **VARYING_STRING**, and is **OPTIONAL**,

28 back - is type default **LOGICAL**, and is **OPTIONAL**,

29 All arguments are scalar.

30 **Action:** The effect of the procedure is to divide the string at the first occurrence of a
31 character that is a member of those included in **set**. The **string** is searched in the
32 forward direction unless **back** is present with the value true, in which case the search is
33 in the backward direction. The characters passed over in the search are returned in the
34 argument **word** and the remainder of the string, not including the separator character is
35 returned in the argument **string**. If no character from **set** is found, the whole string is
36 returned in **word** and **string** is returned as zero-length. If the argument **separator** is
37 present, the actual character found which separates the **word** from the remainder of the
38 **string** is returned in **separator**. The effect of the procedure is such that, on return,
39 either

40 **word//separator//string**

41 is the same as the initial string for a forward search, or

42 **string//separator//word**

43 is the same as the initial string for a backward search.

Annex A

(Informative)

The following module is written in Fortran 90, conformant with the language as specified in the standard ISO/IEC 1539 : 1991. It is intended to be a portable implementation of a module conformant with this International Standard. It is not intended to be prescriptive of how facilities consistent with this International Standard should be provided. This module is intended primarily to demonstrate that portable facilities consistent with the interfaces and semantics required by this International Standard could be provided within the confines of the Fortran language. It is also included as a guide for users of processors which do not have supplier-provided facilities implementing this International Standard.

It should be noted that while every care has been taken by the technical working group to ensure that this module is a correct implementation of this International Standard in valid Fortran code, no guarantee is given or implied that this code will produce correct results, or even that it will execute on any particular processor. Neither is there any implication that this illustrative module is in any way an optimal implementation of this standard; it is merely one fairly straight forward portable module that is known to provide a functionally conformant implementation on a few processors.

```

16  MODULE ISO_VARYING_STRING
17
18  ! Written by J.L.Schonfelder
19  ! Incorporating suggestions by C.Tanasescu, C.Weber, J.Wagener and W.Walter,
20  ! and corrections due to L.Moss, M.Cohen, P.Griffiths, B.T.Smith
21  ! and many other members of the committee ISO/IEC JTC1/SC22/WG5
22
23  ! Version produced (20-Jul-93)
24
25  !-----!
26  ! This module defines the interface and one possible implementation for a      !
27  ! dynamic length character string facility in Fortran 90. The Fortran 90      !
28  ! language is defined by the standard ISO/IEC 1539 : 1991.                    !
29  ! The publicly accessible interface defined by this module is conformant      !
30  ! with the auxilliary standard, ISO/IEC 1539-1 : 1993.                        !
31  ! The detailed implementation may be considered as an informal definition of  !
32  ! the required semantics, and may also be used as a guide to the production  !
33  ! of a portable implementation.                                               !
34  ! N.B. Although every care has been taken to produce valid Fortran code in   !
35  ! construction of this module no guarantee is given or implied that this    !
36  ! code will work correctly without error on any specific processor, nor      !
37  ! is this implementation intended to be in any way optimal either in use     !
38  ! of storage or CPU cycles.                                                  !
39  !-----!
40
41  PRIVATE
42
43  !-----!
44  ! By default all entities declared or defined in this module are private to  !
45  ! the module. Only those entities declared explicitly as being public are     !
46  ! accessible to programs using the module. In particular, the procedures and  !
47  ! operators defined herein are made accessible via their generic identifiers  !
48  ! only; their specific names are private.                                     !
49  !-----!
50
51  TYPE VARYING_STRING
52  PRIVATE
53  CHARACTER, DIMENSION(:), POINTER :: chars
54  ENDTYPE VARYING_STRING
55
56  !-----!
57  ! The representation chosen for this definition of the module is of a string  !
58  ! type consisting of a single component that is a pointer to a rank one array !
59  ! of characters.                                                              !

```

```

1  ! Note: this Module is defined only for characters of default kind. A similar !
2  ! module could be defined for non-default characters if these are supported !
3  ! on a processor by adding a KIND parameter to the component in the type !
4  ! definition, and to all delarations of objects of CHARACTER type. !
5  !-----!
6
7  CHARACTER,PARAMETER :: blank = " "
8
9  !----- GENERIC PROCEDURE INTERFACE DEFINITIONS -----!
10
11  !----- LEN interface -----!
12  INTERFACE LEN
13  MODULE PROCEDURE len_s ! length of string
14  ENDINTERFACE
15
16  !----- Conversion procedure interfaces -----!
17  INTERFACE VAR_STR
18  MODULE PROCEDURE c_to_s ! character to string
19  ENDINTERFACE
20
21  INTERFACE CHAR
22  MODULE PROCEDURE s_to_c, & ! string to character
23  s_to_fix_c ! string to specified length character
24  ENDINTERFACE
25
26  !----- ASSIGNMENT interfaces -----!
27  INTERFACE ASSIGNMENT(=)
28  MODULE PROCEDURE s_ass_s, & ! string = string
29  c_ass_s, & ! character = string
30  s_ass_c ! string = character
31  ENDINTERFACE
32
33  !----- Concatenation operator interfaces -----!
34  INTERFACE OPERATOR(//)
35  MODULE PROCEDURE s_concat_s, & ! string//string
36  s_concat_c, & ! string//character
37  c_concat_s ! character//string
38  ENDINTERFACE
39
40  !----- Repeated Concatenation interfaces -----!
41  INTERFACE REPEAT
42  MODULE PROCEDURE repeat_s
43  ENDINTERFACE
44
45  !----- Equality comparison operator interfaces-----!
46  INTERFACE OPERATOR(==)
47  MODULE PROCEDURE s_eq_s, & ! string==string
48  s_eq_c, & ! string==character
49  c_eq_s ! character==string
50  ENDINTERFACE
51
52  !----- not-equality comparison operator interfaces -----!
53  INTERFACE OPERATOR(/=)
54  MODULE PROCEDURE s_ne_s, & ! string/=string
55  s_ne_c, & ! string/=character
56  c_ne_s ! character/=string
57  ENDINTERFACE
58
59  !----- less-than comparison operator interfaces -----!
60  INTERFACE OPERATOR(<)
61  MODULE PROCEDURE s_lt_s, & ! string<string
62  s_lt_c, & ! string<character
63  c_lt_s ! character<string
64  ENDINTERFACE
65
66  !----- less-than-or-equal comparison operator interfaces -----!
67  INTERFACE OPERATOR(<=)
68  MODULE PROCEDURE s_le_s, & ! string<=string
69  s_le_c, & ! string<=character
70  c_le_s ! character<=string
71  ENDINTERFACE
72

```

```

1  !----- greater-than-or-equal comparison operator interfaces -----!
2  INTERFACE OPERATOR(>=)
3      MODULE PROCEDURE s_ge_s, & ! string>=string
4                          s_ge_c, & ! string>=character
5                          c_ge_s   ! character>=string
6  ENDINTERFACE
7
8  !----- greater-than comparison operator interfaces -----!
9  INTERFACE OPERATOR(>)
10     MODULE PROCEDURE s_gt_s, & ! string>string
11                     s_gt_c, & ! string>character
12                     c_gt_s   ! character>string
13 ENDINTERFACE
14
15 !----- LLT procedure interfaces -----!
16 INTERFACE LLT
17     MODULE PROCEDURE s_llt_s, & ! LLT(string,string)
18                     s_llt_c, & ! LLT(string,character)
19                     c_llt_s   ! LLT(character,string)
20 ENDINTERFACE
21
22 !----- LLE procedure interfaces -----!
23 INTERFACE LLE
24     MODULE PROCEDURE s_lle_s, & ! LLE(string,string)
25                     s_lle_c, & ! LLE(string,character)
26                     c_lle_s   ! LLE(character,string)
27 ENDINTERFACE
28
29 !----- LGE procedure interfaces -----!
30 INTERFACE LGE
31     MODULE PROCEDURE s_lge_s, & ! LGE(string,string)
32                     s_lge_c, & ! LGE(string,character)
33                     c_lge_s   ! LGE(character,string)
34 ENDINTERFACE
35
36 !----- LGT procedure interfaces -----!
37 INTERFACE LGT
38     MODULE PROCEDURE s_lgt_s, & ! LGT(string,string)
39                     s_lgt_c, & ! LGT(string,character)
40                     c_lgt_s   ! LGT(character,string)
41 ENDINTERFACE
42
43 !----- Input function interface -----!
44 INTERFACE GET
45     MODULE PROCEDURE get_d_eor, & ! default unit, EoR termination
46                     get_u_eor, & ! specified unit, EoR termination
47                     get_d_tset_s, & ! default unit, string set termination
48                     get_u_tset_s, & ! specified unit, string set termination
49                     get_d_tset_c, & ! default unit, char set termination
50                     get_u_tset_c   ! specified unit, char set termination
51 ENDINTERFACE
52
53 !----- Output procedure interfaces -----!
54 INTERFACE PUT
55     MODULE PROCEDURE put_d_s, & ! string to default unit
56                     put_u_s, & ! string to specified unit
57                     put_d_c, & ! char to default unit
58                     put_u_c   ! char to specified unit
59 ENDINTERFACE
60
61 INTERFACE PUT_LINE
62     MODULE PROCEDURE putline_d_s, & ! string to default unit
63                     putline_u_s, & ! string to specified unit
64                     putline_d_c, & ! char to default unit
65                     putline_u_c   ! char to specified unit
66 ENDINTERFACE
67
68 !----- Insert procedure interfaces -----!
69 INTERFACE INSERT
70     MODULE PROCEDURE insert_ss, & ! string in string
71                     insert_sc, & ! char in string
72                     insert_cs, & ! string in char

```

```

1          insert_cc    ! char in char
2  ENDINTERFACE

3  !----- Replace procedure interfaces -----!
4  INTERFACE REPLACE
5      MODULE PROCEDURE replace_ss, &    ! string by string, at specified
6          replace_sc, &    ! string by char , starting
7          replace_cs, &    ! char by string , point
8          replace_cc, &    ! char by char
9          replace_ss_sf,& ! string by string, between
10         replace_sc_sf,& ! string by char , specified
11         replace_cs_sf,& ! char by string , starting and
12         replace_cc_sf,& ! char by char , finishing points
13         replace_sss, & ! in string replace string by string
14         replace_ssc, & ! in string replace string by char
15         replace_scs, & ! in string replace char by string
16         replace_scc, & ! in string replace char by char
17         replace_css, & ! in char replace string by string
18         replace_csc, & ! in char replace string by char
19         replace_ccs, & ! in char replace char by string
20         replace_ccc    ! in char replace char by char
21 ENDINTERFACE

22 !----- Remove procedure interface -----!
23 INTERFACE REMOVE
24     MODULE PROCEDURE remove_s, & ! characters from string, between start
25         remove_c    ! characters from char , and finish
26 ENDINTERFACE

27
28 !----- Extract procedure interface -----!
29 INTERFACE EXTRACT
30     MODULE PROCEDURE extract_s, & ! from string extract string, between start
31         extract_c    ! from char  extract string, and finish
32 ENDINTERFACE

33
34 !----- Split procedure interface -----!
35 INTERFACE SPLIT
36     MODULE PROCEDURE split_s, & ! split string at first occurrence of
37         split_c    ! character in set
38 ENDINTERFACE

39
40 !----- Index procedure interfaces -----!
41 INTERFACE INDEX
42     MODULE PROCEDURE index_ss, index_sc, index_cs
43 ENDINTERFACE

44
45 !----- Scan procedure interfaces -----!
46 INTERFACE SCAN
47     MODULE PROCEDURE scan_ss, scan_sc, scan_cs
48 ENDINTERFACE

49
50 !----- Verify procedure interfaces -----!
51 INTERFACE VERIFY
52     MODULE PROCEDURE verify_ss, verify_sc, verify_cs
53 ENDINTERFACE

54
55 !----- Interfaces for remaining intrinsic function overloads -----!
56 INTERFACE LEN_TRIM
57     MODULE PROCEDURE len_trim_s
58 ENDINTERFACE

59 INTERFACE TRIM
60     MODULE PROCEDURE trim_s
61 ENDINTERFACE

62 INTERFACE IACHAR
63     MODULE PROCEDURE iachar_s
64 ENDINTERFACE

65 INTERFACE ICHAR
66     MODULE PROCEDURE ichar_s
67 ENDINTERFACE

```



```

1
2 INTERFACE ADJUSTL
3   MODULE PROCEDURE adjustl_s
4 ENDINTERFACE
5
6 INTERFACE ADJUSTR
7   MODULE PROCEDURE adjustr_s
8 ENDINTERFACE
9
10 !----- specification of publically accessible entities -----!
11 PUBLIC :: VARYING_STRING,VAR_STR,CHAR,LEN,GET,PUT,PUT_LINE,INSERT,REPLACE, &
12          SPLIT,REMOVE,REPEAT,EXTRACT,INDEX,SCAN,VERIFY,LLT,LLE,LGE,LGT, &
13          ASSIGNMENT(=),OPERATOR(/),OPERATOR(==),OPERATOR(/=),OPERATOR(<), &
14          OPERATOR(<=),OPERATOR(>=),OPERATOR(>),LEN_TRIM,TRIM,IACHAR,ICHAR, &
15          ADJUSTL,ADJUSTR
16
17 CONTAINS
18
19 !----- LEN Procedure -----!
20 FUNCTION len_s(string)
21   type(VARYING_STRING),INTENT(IN) :: string
22   INTEGER :: len_s
23   ! returns the length of the string argument or zero if there is no current
24   ! string value
25   IF(.NOT.ASSOCIATED(string%chars))THEN
26     len_s = 0
27   ELSE
28     len_s = SIZE(string%chars)
29   ENDIF
30 ENDFUNCTION len_s
31
32 !----- Conversion Procedures -----!
33 FUNCTION c_to_s(chr)
34   type(VARYING_STRING) :: c_to_s
35   CHARACTER(LEN=*),INTENT(IN) :: chr
36   ! returns the string consisting of the characters char
37   INTEGER :: lc
38   lc=LEN(chr)
39   ALLOCATE(c_to_s%chars(1:lc))
40   DO i=1,lc
41     c_to_s%chars(i) = chr(i:i)
42   ENDDO
43 ENDFUNCTION c_to_s
44
45 FUNCTION s_to_c(string)
46   type(VARYING_STRING),INTENT(IN) :: string
47   CHARACTER(LEN=SIZE(string%chars)) :: s_to_c
48   ! returns the characters of string as an automatically sized character
49   INTEGER :: lc
50   lc=SIZE(string%chars)
51   DO i=1,lc
52     s_to_c(i:i) = string%chars(i)
53   ENDDO
54 ENDFUNCTION s_to_c
55
56 FUNCTION s_to_fix_c(string,length)
57   type(VARYING_STRING),INTENT(IN) :: string
58   INTEGER,INTENT(IN) :: length
59   CHARACTER(LEN=length) :: s_to_fix_c
60   ! returns the character of fixed length, length, containing the characters
61   ! of string either padded with blanks or truncated on the right to fit
62   INTEGER :: lc
63   lc=MIN(SIZE(string%chars),length)
64   DO i=1,lc
65     s_to_fix_c(i:i) = string%chars(i)
66   ENDDO
67   IF(lc < length)THEN ! result longer than string padding needed
68     s_to_fix_c(lc+1:length) = blank
69   ENDIF
70 ENDFUNCTION s_to_fix_c
71
72 !----- ASSIGNMENT Procedures -----!

```

```

1  SUBROUTINE s_ass_s(var,expr)
2  type(VARYING_STRING),INTENT(OUT) :: var
3  type(VARYING_STRING),INTENT(IN)  :: expr
4  ! assign a string value to a string variable overriding default assignment
5  ! reallocates string variable to size of string value and copies characters
6  ALLOCATE(var%chars(1:LEN(expr)))
7  var%chars = expr%chars
8  ENDSUBROUTINE s_ass_s
9
10 SUBROUTINE c_ass_s(var,expr)
11 CHARACTER(LEN=*),INTENT(OUT)  :: var
12 type(VARYING_STRING),INTENT(IN) :: expr
13 ! assign a string value to a character variable
14 ! if the string is longer than the character truncate the string on the right
15 ! if the string is shorter the character is blank padded on the right
16 INTEGER                      :: lc,ls
17 lc = LEN(var); ls = MIN(LEN(expr),lc)
18 DO i = 1,ls
19   var(i:i) = expr%chars(i)
20 ENDDO
21 DO i = ls+1,lc
22   var(i:i) = blank
23 ENDDO
24 ENDSUBROUTINE c_ass_s
25
26 SUBROUTINE s_ass_c(var,expr)
27 type(VARYING_STRING),INTENT(OUT) :: var
28 CHARACTER(LEN=*),INTENT(IN)     :: expr
29 ! assign a character value to a string variable
30 ! disassociates the string variable from its current value, allocates new
31 ! space to hold the characters and copies them from the character value
32 ! into this space.
33 INTEGER                      :: lc
34 lc = LEN(expr)
35 ALLOCATE(var%chars(1:lc))
36 DO i = 1,lc
37   var%chars(i) = expr(i:i)
38 ENDDO
39 ENDSUBROUTINE s_ass_c
40
41 !----- Concatenation operator procedures -----!
42 FUNCTION s_concat_s(string_a,string_b) ! string//string
43 type(VARYING_STRING),INTENT(IN) :: string_a,string_b
44 type(VARYING_STRING)           :: s_concat_s
45 INTEGER                        :: la,lb
46 la = LEN(string_a); lb = LEN(string_b)
47 ALLOCATE(s_concat_s%chars(1:la+lb))
48 s_concat_s%chars(1:la) = string_a%chars
49 s_concat_s%chars(1+la:la+lb) = string_b%chars
50 ENDFUNCTION s_concat_s
51
52 FUNCTION s_concat_c(string_a,string_b) ! string//character
53 type(VARYING_STRING),INTENT(IN) :: string_a
54 CHARACTER(LEN=*),INTENT(IN)     :: string_b
55 type(VARYING_STRING)           :: s_concat_c
56 INTEGER                        :: la,lb
57 la = LEN(string_a); lb = LEN(string_b)
58 ALLOCATE(s_concat_c%chars(1:la+lb))
59 s_concat_c%chars(1:la) = string_a%chars
60 DO i = 1,lb
61   s_concat_c%chars(la+i) = string_b(i:i)
62 ENDDO
63 ENDFUNCTION s_concat_c
64
65 FUNCTION c_concat_s(string_a,string_b) ! character//string
66 CHARACTER(LEN=*),INTENT(IN)     :: string_a
67 type(VARYING_STRING),INTENT(IN) :: string_b
68 type(VARYING_STRING)           :: c_concat_s
69 INTEGER                        :: la,lb
70 la = LEN(string_a); lb = LEN(string_b)
71 ALLOCATE(c_concat_s%chars(1:la+lb))
72 DO i = 1,la

```

```

1      c_concat_s%chars(i) = string_a(i:i)
2      ENDDO
3      c_concat_s%chars(1+la:la+lb) = string_b%chars
4      ENDFUNCTION c_concat_s
5
6  !----- Repeated concatenation procedures -----!
7  FUNCTION repeat_s(string,ncopies)
8      type(VARYING_STRING),INTENT(IN) :: string
9      INTEGER,INTENT(IN)             :: ncopies
10     type(VARYING_STRING)           :: repeat_s
11     ! Returns a string produced by the concatenation of ncopies of the
12     ! argument string
13     INTEGER                         :: lr,ls
14     IF (ncopies < 0) THEN
15         WRITE(*,*) " Negative ncopies requested in REPEAT"
16         STOP
17     ENDIF
18     ls = LEN(string); lr = ls*ncopies
19     ALLOCATE(repeat_s%chars(1:lr))
20     DO i = 1,ncopies
21         repeat_s%chars(1+(i-1)*ls:i*ls) = string%chars
22     ENDDO
23     ENDFUNCTION repeat_s
24
25  !----- Equality comparison operators -----!
26  FUNCTION s_eq_s(string_a,string_b) ! string==string
27     type(VARYING_STRING),INTENT(IN) :: string_a,string_b
28     LOGICAL                          :: s_eq_s
29     INTEGER                           :: la,lb
30     la = LEN(string_a); lb = LEN(string_b)
31     IF (la > lb) THEN
32         s_eq_s = ALL(string_a%chars(1:lb) == string_b%chars) .AND. &
33             ALL(string_a%chars(lb+1:la) == blank)
34     ELSEIF (la < lb) THEN
35         s_eq_s = ALL(string_a%chars == string_b%chars(1:la)) .AND. &
36             ALL(blank == string_b%chars(la+1:lb))
37     ELSE
38         s_eq_s = ALL(string_a%chars == string_b%chars)
39     ENDIF
40     ENDFUNCTION s_eq_s
41
42  FUNCTION s_eq_c(string_a,string_b) ! string==character
43     type(VARYING_STRING),INTENT(IN) :: string_a
44     CHARACTER(LEN=*),INTENT(IN)     :: string_b
45     LOGICAL                          :: s_eq_c
46     INTEGER                           :: la,lb,ls
47     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
48     DO i = 1,ls
49         IF( string_a%chars(i) /= string_b(i:i) )THEN
50             s_eq_c = .FALSE.; RETURN
51         ENDIF
52     ENDDO
53     IF( la > lb .AND. ANY( string_a%chars(lb+1:la) /= blank ) )THEN
54         s_eq_c = .FALSE.; RETURN
55     ELSEIF( la < lb .AND. blank /= string_b(la+1:lb) )THEN
56         s_eq_c = .FALSE.; RETURN
57     ENDIF
58     s_eq_c = .TRUE.
59     ENDFUNCTION s_eq_c
60
61  FUNCTION c_eq_s(string_a,string_b) ! character==string
62     CHARACTER(LEN=*),INTENT(IN)     :: string_a
63     type(VARYING_STRING),INTENT(IN) :: string_b
64     LOGICAL                          :: c_eq_s
65     INTEGER                           :: la,lb,ls
66     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
67     DO i = 1,ls
68         IF( string_a(i:i) /= string_b%chars(i) )THEN
69             c_eq_s = .FALSE.; RETURN
70         ENDIF
71     ENDDO
72     IF( la > lb .AND. string_a(lb+1:la) /= blank )THEN

```

```

1      c_eq_s = .FALSE.; RETURN
2      ELSEIF( la < lb .AND. ANY( blank /= string_b%chars(la+1:lb) ) )THEN
3          c_eq_s = .FALSE.; RETURN
4      ENDIF
5      c_eq_s = .TRUE.
6      ENDFUNCTION c_eq_s
7
8  !----- Non-equality operators -----!
9  FUNCTION s_ne_s(string_a,string_b) ! string/=string
10     type(VARYING_STRING),INTENT(IN) :: string_a,string_b
11     LOGICAL                          :: s_ne_s
12     INTEGER                          :: la,lb
13     la = LEN(string_a); lb = LEN(string_b)
14     IF (la > lb) THEN
15         s_ne_s = ANY(string_a%chars(1:lb) /= string_b%chars) .OR. &
16             ANY(string_a%chars(lb+1:la) /= blank)
17     ELSEIF (la < lb) THEN
18         s_ne_s = ANY(string_a%chars /= string_b%chars(1:la)) .OR. &
19             ANY(blank /= string_b%chars(la+1:lb))
20     ELSE
21         s_ne_s = ANY(string_a%chars /= string_b%chars)
22     ENDIF
23 ENDFUNCTION s_ne_s
24
25 FUNCTION s_ne_c(string_a,string_b) ! string/=character
26     type(VARYING_STRING),INTENT(IN) :: string_a
27     CHARACTER(LEN=*),INTENT(IN)    :: string_b
28     LOGICAL                          :: s_ne_c
29     INTEGER                          :: la,lb,ls
30     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
31     DO i = 1,ls
32         IF( string_a%chars(i) /= string_b(i:i) )THEN
33             s_ne_c = .TRUE.; RETURN
34         ENDIF
35     ENDDO
36     IF( la > lb .AND. ANY( string_a%chars(lb+1:la) /= blank ) )THEN
37         s_ne_c = .TRUE.; RETURN
38     ELSEIF( la < lb .AND. blank /= string_b(la+1:lb) )THEN
39         s_ne_c = .TRUE.; RETURN
40     ENDIF
41     s_ne_c = .FALSE.
42 ENDFUNCTION s_ne_c
43
44 FUNCTION c_ne_s(string_a,string_b) ! character/=string
45     CHARACTER(LEN=*),INTENT(IN)    :: string_a
46     type(VARYING_STRING),INTENT(IN) :: string_b
47     LOGICAL                          :: c_ne_s
48     INTEGER                          :: la,lb,ls
49     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
50     DO i = 1,ls
51         IF( string_a(i:i) /= string_b%chars(i) )THEN
52             c_ne_s = .TRUE.; RETURN
53         ENDIF
54     ENDDO
55     IF( la > lb .AND. string_a(lb+1:la) /= blank )THEN
56         c_ne_s = .TRUE.; RETURN
57     ELSEIF( la < lb .AND. ANY( blank /= string_b%chars(la+1:lb) ) )THEN
58         c_ne_s = .TRUE.; RETURN
59     ENDIF
60     c_ne_s = .FALSE.
61 ENDFUNCTION c_ne_s
62
63 !----- Less-than operators -----!
64 FUNCTION s_lt_s(string_a,string_b) ! string<string
65     type(VARYING_STRING),INTENT(IN) :: string_a,string_b
66     LOGICAL                          :: s_lt_s
67     INTEGER                          :: ls,la,lb
68     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
69     DO i = 1,ls
70         IF( string_a%chars(i) < string_b%chars(i) )THEN
71             s_lt_s = .TRUE.; RETURN
72         ELSEIF( string_a%chars(i) > string_b%chars(i) )THEN

```

```

1      s_lt_s = .FALSE.; RETURN
2      ENDDIF
3      ENDDO
4      IF( la < lb )THEN
5          DO i = la+1,lb
6              IF( blank < string_b%chars(i) )THEN
7                  s_lt_s = .TRUE.; RETURN
8              ELSEIF( blank > string_b%chars(i) )THEN
9                  s_lt_s = .FALSE.; RETURN
10             ENDDIF
11         ENDDO
12     ELSEIF( la > lb )THEN
13         DO i = lb+1,la
14             IF( string_a%chars(i) < blank )THEN
15                 s_lt_s = .TRUE.; RETURN
16             ELSEIF( string_a%chars(i) > blank )THEN
17                 s_lt_s = .FALSE.; RETURN
18             ENDDIF
19         ENDDO
20     ENDDIF
21     s_lt_s = .FALSE.
22 ENDFUNCTION s_lt_s
23
24 FUNCTION s_lt_c(string_a,string_b) ! string<character
25 type(VARYING_STRING),INTENT(IN) :: string_a
26 CHARACTER(LEN=*),INTENT(IN)    :: string_b
27 LOGICAL                        :: s_lt_c
28 INTEGER                        :: ls,la,lb
29 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
30 DO i = 1,ls
31     IF( string_a%chars(i) < string_b(i:i) )THEN
32         s_lt_c = .TRUE.; RETURN
33     ELSEIF( string_a%chars(i) > string_b(i:i) )THEN
34         s_lt_c = .FALSE.; RETURN
35     ENDDIF
36 ENDDO
37 IF( la < lb )THEN
38     IF( blank < string_b(la+1:lb) )THEN
39         s_lt_c = .TRUE.; RETURN
40     ELSEIF( blank > string_b(la+1:lb) )THEN
41         s_lt_c = .FALSE.; RETURN
42     ENDDIF
43 ELSEIF( la > lb )THEN
44     DO i = lb+1,la
45         IF( string_a%chars(i) < blank )THEN
46             s_lt_c = .TRUE.; RETURN
47         ELSEIF( string_a%chars(i) > blank )THEN
48             s_lt_c = .FALSE.; RETURN
49         ENDDIF
50     ENDDO
51 ENDDIF
52 s_lt_c = .FALSE.
53 ENDFUNCTION s_lt_c
54
55 FUNCTION c_lt_s(string_a,string_b) ! character<string
56 CHARACTER(LEN=*),INTENT(IN)    :: string_a
57 type(VARYING_STRING),INTENT(IN) :: string_b
58 LOGICAL                        :: c_lt_s
59 INTEGER                        :: ls,la,lb
60 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
61 DO i = 1,ls
62     IF( string_a(i:i) < string_b%chars(i) )THEN
63         c_lt_s = .TRUE.; RETURN
64     ELSEIF( string_a(i:i) > string_b%chars(i) )THEN
65         c_lt_s = .FALSE.; RETURN
66     ENDDIF
67 ENDDO
68 IF( la < lb )THEN
69     DO i = la+1,lb
70         IF( blank < string_b%chars(i) )THEN
71             c_lt_s = .TRUE.; RETURN
72         ELSEIF( blank > string_b%chars(i) )THEN

```

```

1      c_lt_s = .FALSE.; RETURN
2      ENDIF
3      ENDDO
4      ELSEIF( la > lb )THEN
5          IF( string_a(lb+1:la) < blank )THEN
6              c_lt_s = .TRUE.; RETURN
7          ELSEIF( string_a(lb+1:la) > blank )THEN
8              c_lt_s = .FALSE.; RETURN
9          ENDIF
10     ENDIF
11     c_lt_s = .FALSE.
12 ENDFUNCTION c_lt_s

13 !----- Less-than-or-equal-to operators -----!
14 FUNCTION s_le_s(string_a,string_b) ! string<=string
15     type(VARYING_STRING),INTENT(IN) :: string_a,string_b
16     LOGICAL                          :: s_le_s
17     INTEGER                          :: ls,la,lb
18     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
19     DO i = 1,ls
20         IF( string_a%chars(i) < string_b%chars(i) )THEN
21             s_le_s = .TRUE.; RETURN
22         ELSEIF( string_a%chars(i) > string_b%chars(i) )THEN
23             s_le_s = .FALSE.; RETURN
24         ENDIF
25     ENDDO
26     IF( la < lb )THEN
27         DO i = la+1,lb
28             IF( blank < string_b%chars(i) )THEN
29                 s_le_s = .TRUE.; RETURN
30             ELSEIF( blank > string_b%chars(i) )THEN
31                 s_le_s = .FALSE.; RETURN
32             ENDIF
33         ENDDO
34     ELSEIF( la > lb )THEN
35         DO i = lb+1,la
36             IF( string_a%chars(i) < blank )THEN
37                 s_le_s = .TRUE.; RETURN
38             ELSEIF( string_a%chars(i) > blank )THEN
39                 s_le_s = .FALSE.; RETURN
40             ENDIF
41         ENDDO
42     ENDIF
43     s_le_s = .TRUE.
44 ENDFUNCTION s_le_s

45
46 FUNCTION s_le_c(string_a,string_b) ! string<=character
47     type(VARYING_STRING),INTENT(IN) :: string_a
48     CHARACTER(LEN=*),INTENT(IN)    :: string_b
49     LOGICAL                          :: s_le_c
50     INTEGER                          :: ls,la,lb
51     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
52     DO i = 1,ls
53         IF( string_a%chars(i) < string_b(i:i) )THEN
54             s_le_c = .TRUE.; RETURN
55         ELSEIF( string_a%chars(i) > string_b(i:i) )THEN
56             s_le_c = .FALSE.; RETURN
57         ENDIF
58     ENDDO
59     IF( la < lb )THEN
60         IF( blank < string_b(la+1:lb) )THEN
61             s_le_c = .TRUE.; RETURN
62         ELSEIF( blank > string_b(la+1:lb) )THEN
63             s_le_c = .FALSE.; RETURN
64         ENDIF
65     ELSEIF( la > lb )THEN
66         DO i = lb+1,la
67             IF( string_a%chars(i) < blank )THEN
68                 s_le_c = .TRUE.; RETURN
69             ELSEIF( string_a%chars(i) > blank )THEN
70                 s_le_c = .FALSE.; RETURN
71             ENDIF

```

```

1      ENDDO
2      ENDIF
3      s_le_c = .TRUE.
4      ENDFUNCTION s_le_c
5
6      FUNCTION c_le_s(string_a,string_b) ! character<=string
7      CHARACTER(LEN=*),INTENT(IN)      :: string_a
8      type(VARYING_STRING),INTENT(IN)  :: string_b
9      LOGICAL                          :: c_le_s
10     INTEGER                          :: ls,la,lb
11     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
12     DO i = 1,ls
13         IF( string_a(i:i) < string_b%chars(i) )THEN
14             c_le_s = .TRUE.; RETURN
15         ELSEIF( string_a(i:i) > string_b%chars(i) )THEN
16             c_le_s = .FALSE.; RETURN
17         ENDIF
18     ENDDO
19     IF( la < lb )THEN
20         DO i = la+1,lb
21             IF( blank < string_b%chars(i) )THEN
22                 c_le_s = .TRUE.; RETURN
23             ELSEIF( blank > string_b%chars(i) )THEN
24                 c_le_s = .FALSE.; RETURN
25             ENDIF
26         ENDDO
27     ELSEIF( la > lb )THEN
28         IF( string_a(lb+1:la) < blank )THEN
29             c_le_s = .TRUE.; RETURN
30         ELSEIF( string_a(lb+1:la) > blank )THEN
31             c_le_s = .FALSE.; RETURN
32         ENDIF
33     ENDIF
34     c_le_s = .TRUE.
35     ENDFUNCTION c_le_s
36
37     !----- Greater-than-or-equal-to operators -----!
38     FUNCTION s_ge_s(string_a,string_b) ! string>=string
39     type(VARYING_STRING),INTENT(IN)  :: string_a,string_b
40     LOGICAL                          :: s_ge_s
41     INTEGER                          :: ls,la,lb
42     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
43     DO i = 1,ls
44         IF( string_a%chars(i) > string_b%chars(i) )THEN
45             s_ge_s = .TRUE.; RETURN
46         ELSEIF( string_a%chars(i) < string_b%chars(i) )THEN
47             s_ge_s = .FALSE.; RETURN
48         ENDIF
49     ENDDO
50     IF( la < lb )THEN
51         DO i = la+1,lb
52             IF( blank > string_b%chars(i) )THEN
53                 s_ge_s = .TRUE.; RETURN
54             ELSEIF( blank < string_b%chars(i) )THEN
55                 s_ge_s = .FALSE.; RETURN
56             ENDIF
57         ENDDO
58     ELSEIF( la > lb )THEN
59         DO i = lb+1,la
60             IF( string_a%chars(i) > blank )THEN
61                 s_ge_s = .TRUE.; RETURN
62             ELSEIF( string_a%chars(i) < blank )THEN
63                 s_ge_s = .FALSE.; RETURN
64             ENDIF
65         ENDDO
66     ENDIF
67     s_ge_s = .TRUE.
68     ENDFUNCTION s_ge_s
69
70     FUNCTION s_ge_c(string_a,string_b) ! string>=character
71     type(VARYING_STRING),INTENT(IN)  :: string_a
72     CHARACTER(LEN=*),INTENT(IN)      :: string_b

```

```

1      LOGICAL                :: s_ge_c
2      INTEGER                :: ls,la,lb
3      la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
4      DO i = 1,ls
5          IF( string_a%chars(i) > string_b(i:i) )THEN
6              s_ge_c = .TRUE.; RETURN
7          ELSEIF( string_a%chars(i) < string_b(i:i) )THEN
8              s_ge_c = .FALSE.; RETURN
9          ENDIF
10     ENDDO
11     IF( la < lb )THEN
12         IF( blank > string_b(la+1:lb) )THEN
13             s_ge_c = .TRUE.; RETURN
14         ELSEIF( blank < string_b(la+1:lb) )THEN
15             s_ge_c = .FALSE.; RETURN
16         ENDIF
17     ELSEIF( la > lb )THEN
18         DO i = lb+1,la
19             IF( string_a%chars(i) > blank )THEN
20                 s_ge_c = .TRUE.; RETURN
21             ELSEIF( string_a%chars(i) < blank )THEN
22                 s_ge_c = .FALSE.; RETURN
23             ENDIF
24         ENDDO
25     ENDIF
26     s_ge_c = .TRUE.
27 ENDFUNCTION s_ge_c
28
29 FUNCTION c_ge_s(string_a,string_b) ! character>=string
30 CHARACTER(LEN=*) ,INTENT(IN)    :: string_a
31 type(VARYING_STRING),INTENT(IN) :: string_b
32 LOGICAL                          :: c_ge_s
33 INTEGER                          :: ls,la,lb
34 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
35 DO i = 1,ls
36     IF( string_a(i:i) > string_b%chars(i) )THEN
37         c_ge_s = .TRUE.; RETURN
38     ELSEIF( string_a(i:i) < string_b%chars(i) )THEN
39         c_ge_s = .FALSE.; RETURN
40     ENDIF
41 ENDDO
42 IF( la < lb )THEN
43     DO i = la+1,lb
44         IF( blank > string_b%chars(i) )THEN
45             c_ge_s = .TRUE.; RETURN
46         ELSEIF( blank < string_b%chars(i) )THEN
47             c_ge_s = .FALSE.; RETURN
48         ENDIF
49     ENDDO
50 ELSEIF( la > lb )THEN
51     IF( string_a(lb+1:la) > blank )THEN
52         c_ge_s = .TRUE.; RETURN
53     ELSEIF( string_a(lb+1:la) < blank )THEN
54         c_ge_s = .FALSE.; RETURN
55     ENDIF
56 ENDIF
57 c_ge_s = .TRUE.
58 ENDFUNCTION c_ge_s
59
60 !----- Greater-than operators -----!
61 FUNCTION s_gt_s(string_a,string_b) ! string>string
62 type(VARYING_STRING),INTENT(IN) :: string_a,string_b
63 LOGICAL                          :: s_gt_s
64 INTEGER                          :: ls,la,lb
65 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
66 DO i = 1,ls
67     IF( string_a%chars(i) > string_b%chars(i) )THEN
68         s_gt_s = .TRUE.; RETURN
69     ELSEIF( string_a%chars(i) < string_b%chars(i) )THEN
70         s_gt_s = .FALSE.; RETURN
71     ENDIF
72 ENDDO

```



```

1  IF( la < lb )THEN
2  DO i = la+1,lb
3  IF( blank > string_b%chars(i) )THEN
4  s_gt_s = .TRUE.; RETURN
5  ELSEIF( blank < string_b%chars(i) )THEN
6  s_gt_s = .FALSE.; RETURN
7  ENDIF
8  ENDDO
9  ELSEIF( la > lb )THEN
10 DO i = lb+1,la
11 IF( string_a%chars(i) > blank )THEN
12 s_gt_s = .TRUE.; RETURN
13 ELSEIF( string_a%chars(i) < blank )THEN
14 s_gt_s = .FALSE.; RETURN
15 ENDIF
16 ENDDO
17 ENDIF
18 s_gt_s = .FALSE.
19 ENDFUNCTION s_gt_s
20
21 FUNCTION s_gt_c(string_a,string_b) ! string>character
22 type(VARYING_STRING),INTENT(IN) :: string_a
23 CHARACTER(LEN=*),INTENT(IN)    :: string_b
24 LOGICAL                        :: s_gt_c
25 INTEGER                        :: ls,la,lb
26 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
27 DO i = 1,ls
28 IF( string_a%chars(i) > string_b(i:i) )THEN
29 s_gt_c = .TRUE.; RETURN
30 ELSEIF( string_a%chars(i) < string_b(i:i) )THEN
31 s_gt_c = .FALSE.; RETURN
32 ENDIF
33 ENDDO
34 IF( la < lb )THEN
35 IF( blank > string_b(la+1:lb) )THEN
36 s_gt_c = .TRUE.; RETURN
37 ELSEIF( blank < string_b(la+1:lb) )THEN
38 s_gt_c = .FALSE.; RETURN
39 ENDIF
40 ELSEIF( la > lb )THEN
41 DO i = lb+1,la
42 IF( string_a%chars(i) > blank )THEN
43 s_gt_c = .TRUE.; RETURN
44 ELSEIF( string_a%chars(i) < blank )THEN
45 s_gt_c = .FALSE.; RETURN
46 ENDIF
47 ENDDO
48 ENDIF
49 s_gt_c = .FALSE.
50 ENDFUNCTION s_gt_c
51
52 FUNCTION c_gt_s(string_a,string_b) ! character>string
53 CHARACTER(LEN=*),INTENT(IN)    :: string_a
54 type(VARYING_STRING),INTENT(IN) :: string_b
55 LOGICAL                        :: c_gt_s
56 INTEGER                        :: ls,la,lb
57 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
58 DO i = 1,ls
59 IF( string_a(i:i) > string_b%chars(i) )THEN
60 c_gt_s = .TRUE.; RETURN
61 ELSEIF( string_a(i:i) < string_b%chars(i) )THEN
62 c_gt_s = .FALSE.; RETURN
63 ENDIF
64 ENDDO
65 IF( la < lb )THEN
66 DO i = la+1,lb
67 IF( blank > string_b%chars(i) )THEN
68 c_gt_s = .TRUE.; RETURN
69 ELSEIF( blank < string_b%chars(i) )THEN
70 c_gt_s = .FALSE.; RETURN
71 ENDIF
72 ENDDO

```

```

1      ELSEIF( la > lb )THEN
2          IF( string_a(lb+1:la) > blank )THEN
3              c_gt_s = .TRUE.; RETURN
4          ELSEIF( string_a(lb+1:la) < blank )THEN
5              c_gt_s = .FALSE.; RETURN
6          ENDIF
7      ENDIF
8      c_gt_s = .FALSE.
9  ENDFUNCTION c_gt_s
10
11  !----- LLT procedures -----!
12  FUNCTION s_llt_s(string_a,string_b) ! string_a<string_b ISO-646 ordering
13  type(VARYING_STRING),INTENT(IN) :: string_a,string_b
14  LOGICAL :: s_llt_s
15  ! Returns TRUE if string_a precedes string_b in the ISO 646 collating
16  ! sequence. Otherwise the result is FALSE. The result is FALSE if both
17  ! string_a and string_b are zero length.
18  INTEGER :: ls,la,lb
19  la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
20  DO i = 1,ls
21      IF( LLT(string_a%chars(i),string_b%chars(i)) )THEN
22          s_llt_s = .TRUE.; RETURN
23      ELSEIF( LGT(string_a%chars(i),string_b%chars(i)) )THEN
24          s_llt_s = .FALSE.; RETURN
25      ENDIF
26  ENDDO
27  IF( la < lb )THEN
28      DO i = la+1,lb
29          IF( LLT(blank,string_b%chars(i)) )THEN
30              s_llt_s = .TRUE.; RETURN
31          ELSEIF( LGT(blank,string_b%chars(i)) )THEN
32              s_llt_s = .FALSE.; RETURN
33          ENDIF
34      ENDDO
35  ELSEIF( la > lb )THEN
36      DO i = lb+1,la
37          IF( LLT(string_a%chars(i),blank) )THEN
38              s_llt_s = .TRUE.; RETURN
39          ELSEIF( LGT(string_a%chars(i),blank) )THEN
40              s_llt_s = .FALSE.; RETURN
41          ENDIF
42      ENDDO
43  ENDIF
44  s_llt_s = .FALSE.
45  ENDFUNCTION s_llt_s
46
47  FUNCTION s_llt_c(string_a,string_b)
48  type(VARYING_STRING),INTENT(IN) :: string_a
49  CHARACTER(LEN=*),INTENT(IN) :: string_b
50  LOGICAL :: s_llt_c
51  INTEGER :: ls,la,lb
52  la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
53  DO i = 1,ls
54      IF( LLT(string_a%chars(i),string_b(i:i)) )THEN
55          s_llt_c = .TRUE.; RETURN
56      ELSEIF( LGT(string_a%chars(i),string_b(i:i)) )THEN
57          s_llt_c = .FALSE.; RETURN
58      ENDIF
59  ENDDO
60  IF( la < lb )THEN
61      IF( LLT(blank,string_b(la+1:lb)) )THEN
62          s_llt_c = .TRUE.; RETURN
63      ELSEIF( LGT(blank,string_b(la+1:lb)) )THEN
64          s_llt_c = .FALSE.; RETURN
65      ENDIF
66  ELSEIF( la > lb )THEN
67      DO i = lb+1,la
68          IF( LLT(string_a%chars(i),blank) )THEN
69              s_llt_c = .TRUE.; RETURN
70          ELSEIF( LGT(string_a%chars(i),blank) )THEN
71              s_llt_c = .FALSE.; RETURN
72      ENDIF

```

```

1      ENDDO
2      ENDF
3      s_llt_c = .FALSE.
4      ENDFUNCTION s_llt_c
5
6      FUNCTION c_llt_s(string_a,string_b) ! string_a,string_b ISO-646 ordering
7      CHARACTER(LEN=*),INTENT(IN)      :: string_a
8      type(VARYING_STRING),INTENT(IN)  :: string_b
9      LOGICAL                          :: c_llt_s
10     INTEGER                          :: ls,la,lb
11     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
12     DO i = 1,ls
13         IF( LLT(string_a(i:i),string_b%chars(i)) )THEN
14             c_llt_s = .TRUE.; RETURN
15         ELSEIF( LGT(string_a(i:i),string_b%chars(i)) )THEN
16             c_llt_s = .FALSE.; RETURN
17         ENDIF
18     ENDDO
19     IF( la < lb )THEN
20         DO i = la+1,lb
21             IF( LLT(blank,string_b%chars(i)) )THEN
22                 c_llt_s = .TRUE.; RETURN
23             ELSEIF( LGT(blank,string_b%chars(i)) )THEN
24                 c_llt_s = .FALSE.; RETURN
25             ENDIF
26         ENDDO
27     ELSEIF( la > lb )THEN
28         IF( LLT(string_a(lb+1:la),blank) )THEN
29             c_llt_s = .TRUE.; RETURN
30         ELSEIF( LGT(string_a(lb+1:la),blank) )THEN
31             c_llt_s = .FALSE.; RETURN
32         ENDIF
33     ENDIF
34     c_llt_s = .FALSE.
35     ENDFUNCTION c_llt_s
36
37     !----- LLE procedures -----!
38     FUNCTION s_lle_s(string_a,string_b) ! string_a<=string_b ISO-646 ordering
39     type(VARYING_STRING),INTENT(IN)  :: string_a,string_b
40     LOGICAL                          :: s_lle_s
41     ! Returns TRUE if strings are equal or if string_a preceeds string_b in the
42     ! ISO 646 collating sequence. Otherwise the result is FALSE.
43     INTEGER                          :: ls,la,lb
44     la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
45     DO i = 1,ls
46         IF( LLT(string_a%chars(i),string_b%chars(i)) )THEN
47             s_lle_s = .TRUE.; RETURN
48         ELSEIF( LGT(string_a%chars(i),string_b%chars(i)) )THEN
49             s_lle_s = .FALSE.; RETURN
50         ENDIF
51     ENDDO
52     IF( la < lb )THEN
53         DO i = la+1,lb
54             IF( LLT(blank,string_b%chars(i)) )THEN
55                 s_lle_s = .TRUE.; RETURN
56             ELSEIF( LGT(blank,string_b%chars(i)) )THEN
57                 s_lle_s = .FALSE.; RETURN
58             ENDIF
59         ENDDO
60     ELSEIF( la > lb )THEN
61         DO i = lb+1,la
62             IF( LLT(string_a%chars(i),blank) )THEN
63                 s_lle_s = .TRUE.; RETURN
64             ELSEIF( LGT(string_a%chars(i),blank) )THEN
65                 s_lle_s = .FALSE.; RETURN
66             ENDIF
67         ENDDO
68     ENDIF
69     s_lle_s = .TRUE.
70     ENDFUNCTION s_lle_s
71
72     FUNCTION s_lle_c(string_a,string_b) ! strung_a<=string_b ISO-646 ordering

```

```

1  type(VARYING_STRING),INTENT(IN) :: string_a
2  CHARACTER(LEN=*),INTENT(IN)    :: string_b
3  LOGICAL                          :: s_lle_c
4  INTEGER                          :: ls,la,lb
5  la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
6  DO i = 1,ls
7      IF( LLT(string_a%chars(i),string_b(i:i)) )THEN
8          s_lle_c = .TRUE.; RETURN
9      ELSEIF( LGT(string_a%chars(i),string_b(i:i)) )THEN
10         s_lle_c = .FALSE.; RETURN
11     ENDIF
12 ENDDO
13 IF( la < lb )THEN
14     IF( LLT(blank,string_b(la+1:lb)) )THEN
15         s_lle_c = .TRUE.; RETURN
16     ELSEIF( LGT(blank,string_b(la+1:lb)) )THEN
17         s_lle_c = .FALSE.; RETURN
18     ENDIF
19 ELSEIF( la > lb )THEN
20     DO i = lb+1,la
21         IF( LLT(string_a%chars(i),blank) )THEN
22             s_lle_c = .TRUE.; RETURN
23         ELSEIF( LGT(string_a%chars(i),blank) )THEN
24             s_lle_c = .FALSE.; RETURN
25         ENDIF
26     ENDDO
27 ENDIF
28 s_lle_c = .TRUE.
29 ENDFUNCTION s_lle_c
30
31 FUNCTION c_lle_s(string_a,string_b) ! string_a<=string_b ISO-646 ordering
32 CHARACTER(LEN=*),INTENT(IN)    :: string_a
33 type(VARYING_STRING),INTENT(IN) :: string_b
34 LOGICAL                          :: c_lle_s
35 INTEGER                          :: ls,la,lb
36 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
37 DO i = 1,ls
38     IF( LLT(string_a(i:i),string_b%chars(i)) )THEN
39         c_lle_s = .TRUE.; RETURN
40     ELSEIF( LGT(string_a(i:i),string_b%chars(i)) )THEN
41         c_lle_s = .FALSE.; RETURN
42     ENDIF
43 ENDDO
44 IF( la < lb )THEN
45     DO i = la+1,lb
46         IF( LLT(blank,string_b%chars(i)) )THEN
47             c_lle_s = .TRUE.; RETURN
48         ELSEIF( LGT(blank,string_b%chars(i)) )THEN
49             c_lle_s = .FALSE.; RETURN
50         ENDIF
51     ENDDO
52 ELSEIF( la > lb )THEN
53     IF( LLT(string_a(lb+1:la),blank) )THEN
54         c_lle_s = .TRUE.; RETURN
55     ELSEIF( LGT(string_a(lb+1:la),blank) )THEN
56         c_lle_s = .FALSE.; RETURN
57     ENDIF
58 ENDIF
59 c_lle_s = .TRUE.
60 ENDFUNCTION c_lle_s
61
62 !----- LGE procedures -----!
63 FUNCTION s_lge_s(string_a,string_b) ! string_a>=string_b ISO-646 ordering
64 type(VARYING_STRING),INTENT(IN) :: string_a,string_b
65 LOGICAL                          :: s_lge_s
66 ! Returns TRUE if strings are equal or if string_a follows string_b in the
67 ! ISO 646 collating sequence. Otherwise the result is FALSE.
68 INTEGER                          :: ls,la,lb
69 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
70 DO i = 1,ls
71     IF( LGT(string_a%chars(i),string_b%chars(i)) )THEN
72         s_lge_s = .TRUE.; RETURN

```

```

1      ELSEIF( LLT(string_a%chars(i),string_b%chars(i)) )THEN
2          s_lge_s = .FALSE.; RETURN
3      ENDIF
4  ENDDO
5  IF( la < lb )THEN
6      DO i = la+1,lb
7          IF( LGT(blank,string_b%chars(i)) )THEN
8              s_lge_s = .TRUE.; RETURN
9          ELSEIF( LLT(blank,string_b%chars(i)) )THEN
10             s_lge_s = .FALSE.; RETURN
11         ENDIF
12     ENDDO
13 ELSEIF( la > lb )THEN
14     DO i = lb+1,la
15         IF( LGT(string_a%chars(i),blank) )THEN
16             s_lge_s = .TRUE.; RETURN
17         ELSEIF( LLT(string_a%chars(i),blank) )THEN
18             s_lge_s = .FALSE.; RETURN
19         ENDIF
20     ENDDO
21 ENDIF
22 s_lge_s = .TRUE.
23 ENDFUNCTION s_lge_s
24
25 FUNCTION s_lge_c(string_a,string_b) ! string_a>=string_b ISO-646 ordering
26 type(VARYING_STRING),INTENT(IN) :: string_a
27 CHARACTER(LEN=*),INTENT(IN)    :: string_b
28 LOGICAL                          :: s_lge_c
29 INTEGER                          :: ls,la,lb
30 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
31 DO i = 1,ls
32     IF( LGT(string_a%chars(i),string_b(i:i)) )THEN
33         s_lge_c = .TRUE.; RETURN
34     ELSEIF( LLT(string_a%chars(i),string_b(i:i)) )THEN
35         s_lge_c = .FALSE.; RETURN
36     ENDIF
37 ENDDO
38 IF( la < lb )THEN
39     IF( LGT(blank,string_b(la+1:lb)) )THEN
40         s_lge_c = .TRUE.; RETURN
41     ELSEIF( LLT(blank,string_b(la+1:lb)) )THEN
42         s_lge_c = .FALSE.; RETURN
43     ENDIF
44 ELSEIF( la > lb )THEN
45     DO i = lb+1,la
46         IF( LGT(string_a%chars(i),blank) )THEN
47             s_lge_c = .TRUE.; RETURN
48         ELSEIF( LLT(string_a%chars(i),blank) )THEN
49             s_lge_c = .FALSE.; RETURN
50         ENDIF
51     ENDDO
52 ENDIF
53 s_lge_c = .TRUE.
54 ENDFUNCTION s_lge_c
55
56 FUNCTION c_lge_s(string_a,string_b) ! string_a>=string_b ISO-646 ordering
57 CHARACTER(LEN=*),INTENT(IN)    :: string_a
58 type(VARYING_STRING),INTENT(IN) :: string_b
59 LOGICAL                          :: c_lge_s
60 INTEGER                          :: ls,la,lb
61 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
62 DO i = 1,ls
63     IF( LGT(string_a(i:i),string_b%chars(i)) )THEN
64         c_lge_s = .TRUE.; RETURN
65     ELSEIF( LLT(string_a(i:i),string_b%chars(i)) )THEN
66         c_lge_s = .FALSE.; RETURN
67     ENDIF
68 ENDDO
69 IF( la < lb )THEN
70     DO i = la+1,lb
71         IF( LGT(blank,string_b%chars(i)) )THEN
72             c_lge_s = .TRUE.; RETURN

```

```

1         ELSEIF( LLT(blank,string_b%chars(i)) )THEN
2           c_lge_s = .FALSE.; RETURN
3         ENDIF
4       ENDDO
5     ELSEIF( la > lb )THEN
6       IF( LGT(string_a(lb+1:la),blank) )THEN
7         c_lge_s = .TRUE.; RETURN
8       ELSEIF( LLT(string_a(lb+1:la),blank) )THEN
9         c_lge_s = .FALSE.; RETURN
10      ENDIF
11    ENDIF
12    c_lge_s = .TRUE.
13  ENDFUNCTION c_lge_s

14  !----- LGT procedures -----!
15  FUNCTION s_lgt_s(string_a,string_b) ! string_a>string_b ISO-646 ordering
16    type(VARYING_STRING),INTENT(IN) :: string_a,string_b
17    LOGICAL                          :: s_lgt_s
18    ! Returns TRUE if string_a follows string_b in the ISO 646 collating sequence.
19    ! Otherwise the result is FALSE. The result is FALSE if both string_a and
20    ! string_b are zero length.
21    INTEGER                          :: ls,la,lb
22    la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
23    DO i = 1,ls
24      IF( LGT(string_a%chars(i),string_b%chars(i)) )THEN
25        s_lgt_s = .TRUE.; RETURN
26      ELSEIF( LLT(string_a%chars(i),string_b%chars(i)) )THEN
27        s_lgt_s = .FALSE.; RETURN
28      ENDIF
29    ENDDO
30    IF( la < lb )THEN
31      DO i = la+1,lb
32        IF( LGT(blank,string_b%chars(i)) )THEN
33          s_lgt_s = .TRUE.; RETURN
34        ELSEIF( LLT(blank,string_b%chars(i)) )THEN
35          s_lgt_s = .FALSE.; RETURN
36        ENDIF
37      ENDDO
38    ELSEIF( la > lb )THEN
39      DO i = lb+1,la
40        IF( LGT(string_a%chars(i),blank) )THEN
41          s_lgt_s = .TRUE.; RETURN
42        ELSEIF( LLT(string_a%chars(i),blank) )THEN
43          s_lgt_s = .FALSE.; RETURN
44        ENDIF
45      ENDDO
46    ENDIF
47    s_lgt_s = .FALSE.
48  ENDFUNCTION s_lgt_s

49
50  FUNCTION s_lgt_c(string_a,string_b) ! string_a>string_b ISO-646 ordering
51    type(VARYING_STRING),INTENT(IN) :: string_a
52    CHARACTER(LEN=*),INTENT(IN)    :: string_b
53    LOGICAL                          :: s_lgt_c
54    INTEGER                          :: ls,la,lb
55    la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
56    DO i = 1,ls
57      IF( LGT(string_a%chars(i),string_b(i:i)) )THEN
58        s_lgt_c = .TRUE.; RETURN
59      ELSEIF( LLT(string_a%chars(i),string_b(i:i)) )THEN
60        s_lgt_c = .FALSE.; RETURN
61      ENDIF
62    ENDDO
63    IF( la < lb )THEN
64      IF( LGT(blank,string_b(la+1:lb)) )THEN
65        s_lgt_c = .TRUE.; RETURN
66      ELSEIF( LLT(blank,string_b(la+1:lb)) )THEN
67        s_lgt_c = .FALSE.; RETURN
68      ENDIF
69    ELSEIF( la > lb )THEN
70      DO i = lb+1,la
71        IF( LGT(string_a%chars(i),blank) )THEN

```

```

1      s_lgt_c = .TRUE.; RETURN
2      ELSEIF( LLT(string_a%chars(i),blank) )THEN
3          s_lgt_c = .FALSE.; RETURN
4      ENDIF
5      ENDDO
6      ENDIF
7      s_lgt_c = .FALSE.
8  ENDFUNCTION s_lgt_c
9
10 FUNCTION c_lgt_s(string_a,string_b) ! string_a>string_b ISO-646 ordering
11 CHARACTER(LEN=*) ,INTENT(IN)      :: string_a
12 type(VARYING_STRING),INTENT(IN) :: string_b
13 LOGICAL                          :: c_lgt_s
14 INTEGER                          :: ls,la,lb
15 la = LEN(string_a); lb = LEN(string_b); ls = MIN(la,lb)
16 DO i = 1,ls
17     IF( LGT(string_a(i:i),string_b%chars(i)) )THEN
18         c_lgt_s = .TRUE.; RETURN
19     ELSEIF( LLT(string_a(i:i),string_b%chars(i)) )THEN
20         c_lgt_s = .FALSE.; RETURN
21     ENDIF
22 ENDDO
23 IF( la < lb )THEN
24     DO i = la+1,lb
25         IF( LGT(blank,string_b%chars(i)) )THEN
26             c_lgt_s = .TRUE.; RETURN
27         ELSEIF( LLT(blank,string_b%chars(i)) )THEN
28             c_lgt_s = .FALSE.; RETURN
29         ENDIF
30     ENDDO
31 ELSEIF( la > lb )THEN
32     IF( LGT(string_a(lb+1:la),blank) )THEN
33         c_lgt_s = .TRUE.; RETURN
34     ELSEIF( LLT(string_a(lb+1:la),blank) )THEN
35         c_lgt_s = .FALSE.; RETURN
36     ENDIF
37 ENDIF
38 c_lgt_s = .FALSE.
39 ENDFUNCTION c_lgt_s
40
41
42 !----- Input string procedure -----!
43 SUBROUTINE get_d_eor(string,maxlen,iostat)
44 type(VARYING_STRING),INTENT(OUT) :: string
45                                     ! the string variable to be filled with
46                                     ! characters read from the
47                                     ! file connected to the default unit
48 INTEGER,INTENT(IN),OPTIONAL        :: maxlen
49                                     ! if present indicates the maximum
50                                     ! number of characters that will be
51                                     ! read from the file
52 INTEGER,INTENT(OUT),OPTIONAL        :: iostat
53                                     ! if present used to return the status
54                                     ! of the data transfer
55                                     ! if absent errors cause termination
56 ! reads string from the default unit starting at next character in the file
57 ! and terminating at the end of record or after maxlen characters.
58 CHARACTER(LEN=80) :: buffer
59 INTEGER           :: ist,nch,toread,nb
60 IF(PRESENT(maxlen))THEN
61     toread=maxlen
62 ELSE
63     toread=HUGE(1)
64 ENDIF
65 string = "" ! clears return string
66 DO ! repeatedly read buffer and add to string until EoR
67     ! or maxlen reached
68     IF(toread <= 0)EXIT
69     nb=MIN(80,toread)
70     READ(*,FMT='(A)',ADVANCE='NO',EOR=9999,SIZE=nch,IOSTAT=ist) buffer(1:nb)
71     IF( ist /= 0 )THEN
72         IF(PRESENT(iostat)) THEN

```

```

1       iostat=ist
2       RETURN
3       ELSE
4         WRITE(*,*) " Error No.",ist, &
5           " during READ_STRING of varying string on default unit"
6       STOP
7       ENDIF
8     ENDIF
9     string = string //buffer(1:nb)
10    toread = toread - nb
11  ENDDO
12  IF(PRESENT(iostat)) iostat = 0
13  RETURN
14  9999 string = string //buffer(1:nch)
15  IF(PRESENT(iostat)) iostat = ist
16  ENDSUBROUTINE get_d_eor
17
18  SUBROUTINE get_u_eor(unit,string,maxlen,iostat)
19    INTEGER,INTENT(IN)      :: unit
20                          ! identifies the input unit which must be
21                          ! connected for sequential formatted read
22    type(VARYING_STRING),INTENT(OUT) :: string
23                          ! the string variable to be filled with
24                          ! characters read from the
25                          ! file connected to the unit
26    INTEGER,INTENT(IN),OPTIONAL :: maxlen
27                          ! if present indicates the maximum
28                          ! number of characters that will be
29                          ! read from the file
30    INTEGER,INTENT(OUT),OPTIONAL :: iostat
31                          ! if present used to return the status
32                          ! of the data transfer
33                          ! if absent errors cause termination
34  ! reads string from unit starting at next character in the file and
35  ! terminating at the end of record or after maxlen characters.
36  CHARACTER(LEN=80) :: buffer
37  INTEGER           :: ist,nch,toread,nb
38  IF(PRESENT(maxlen))THEN
39    toread=maxlen
40  ELSE
41    toread=HUGE(1)
42  ENDIF
43  string="" ! clears return string
44  DO ! repeatedly read buffer and add to string until Eor
45    ! or maxlen reached
46    IF(toread <= 0)EXIT
47    nb=MIN(80,toread)
48    READ(unit,FMT='(A)',ADVANCE='NO',EOR=9999,SIZE=nch,IOSTAT=ist) buffer(1:nb)
49    IF( ist /= 0 )THEN
50      IF(PRESENT(iostat)) THEN
51        iostat=ist
52        RETURN
53      ELSE
54        WRITE(*,*) " Error No.",ist, &
55          " during READ_STRING of varying string on UNIT ",unit
56      STOP
57      ENDIF
58    ENDIF
59    string = string //buffer(1:nb)
60    toread = toread - nb
61  ENDDO
62  IF(PRESENT(iostat)) iostat = 0
63  RETURN
64  9999 string = string //buffer(1:nch)
65  IF(PRESENT(iostat)) iostat = ist
66  ENDSUBROUTINE get_u_eor
67
68  SUBROUTINE get_d_tset_s(string,set,maxlen,iostat)
69    type(VARYING_STRING),INTENT(OUT) :: string
70                          ! the string variable to be filled with
71                          ! characters read from the
72                          ! file connected to the default unit

```



```

1      type(VARYING_STRING),INTENT(IN)  :: set
2      ! the set of characters which if found in
3      ! the input terminate the read
4      INTEGER,INTENT(IN),OPTIONAL     :: maxlen
5      ! if present indicates the maximum
6      ! number of characters that will be
7      ! read from the file
8      INTEGER,INTENT(OUT),OPTIONAL    :: iostat
9      ! if present used to return the status
10     ! of the data transfer
11     ! if absent errors cause termination
12     ! reads string from the default unit starting at next character in the file and
13     ! terminating at the end of record, occurrence of a character in set,
14     ! or after reading maxlen characters.
15     CHARACTER :: buffer ! characters must be read one at a time to detect
16     ! first terminator character in set
17     INTEGER   :: ist,toread,lenset
18     ist=0
19     lenset = LEN(set)
20     IF(PRESENT(maxlen))THEN
21         toread=maxlen
22     ELSE
23         toread=HUGE(1)
24     ENDIF
25     string = "" ! clears return string
26     DO ! repeatedly read buffer and add to string until EoR
27         ! or maxlen reached
28         IF(toread <= 0)EXIT
29         READ(*,FMT='(A)',ADVANCE='NO',EOR=9999,IOSTAT=ist) buffer
30         IF(ist /= 0)THEN
31             IF(PRESENT(iostat)) THEN
32                 iostat=ist
33                 RETURN
34             ELSE
35                 WRITE(*,*) " Error No.",ist, &
36                 " during READ_STRING of varying string on default unit"
37                 STOP
38             ENDIF
39         ENDIF
40         ! check for occurrence of set character in buffer
41         DO j = 1,lenset
42             IF(buffer == set%chars(j)) GOTO 9999
43         ENDDO
44         string = string//buffer
45         toread = toread - 1
46     ENDDO
47     IF(PRESENT(iostat)) iostat = ist
48     RETURN
49     9999 string = string//buffer
50     IF(PRESENT(iostat)) iostat = ist
51     ENDSUBROUTINE get_d_tset_s

52     SUBROUTINE get_u_tset_s(unit,string,set,maxlen,iostat)
53     INTEGER,INTENT(IN)
54     :: unit
55     ! identifies the input unit which must be
56     ! connected for sequential formatted read
57     type(VARYING_STRING),INTENT(OUT) :: string
58     ! the string variable to be filled with
59     ! characters read from the
60     ! file connected to the unit
61     type(VARYING_STRING),INTENT(IN)  :: set
62     ! the set of characters which if found in
63     ! the input terminate the read
64     INTEGER,INTENT(IN),OPTIONAL     :: maxlen
65     ! if present indicates the maximum
66     ! number of characters that will be
67     ! read from the file
68     INTEGER,INTENT(OUT),OPTIONAL    :: iostat
69     ! if present used to return the status
70     ! of the data transfer
71     ! if absent errors cause termination
72     ! reads string from unit starting at next character in the file and

```

```

1  ! terminating at the end of record, occurrence of a character in set,
2  ! or after reading maxlen characters.
3  CHARACTER :: buffer ! characters must be read one at a time to detect
4                ! first terminator character in set
5  INTEGER   :: ist,toread,lenset
6  ist=0
7  lenset = LEN(set)
8  IF(PRESENT(maxlen))THEN
9      toread=maxlen
10 ELSE
11     toread=HUGE(1)
12 ENDIF
13 string = "" ! clears return string
14 DO ! repeatedly read buffer and add to string until EoR
15     ! or maxlen reached
16     IF(toread <= 0)EXIT
17     READ(unit,FMT='(A)',ADVANCE='NO',EOR=9999,IOSTAT=ist) buffer
18     IF(ist /= 0)THEN
19         IF(PRESENT(iostat)) THEN
20             iostat=ist
21             RETURN
22         ELSE
23             WRITE(*,*) " Error No.",ist, &
24                 " during READ_STRING of varying string on unit ",unit
25             STOP
26         ENDIF
27     ENDIF
28     ! check for occurrence of set character in buffer
29     DO j = 1,lenset
30         IF(buffer == set%chars(j)) GOTO 9999
31     ENDDO
32     string = string//buffer
33     toread = toread - 1
34 ENDDO
35 IF(PRESENT(iostat)) iostat = ist
36 RETURN
37 9999 string = string//buffer
38 IF(PRESENT(iostat)) iostat = ist
39 ENDSUBROUTINE get_u_tset_s

40 SUBROUTINE get_d_tset_c(string,set,maxlen,iostat)
41     type(VARYING_STRING),INTENT(OUT) :: string
42                                     ! the string variable to be filled with
43                                     ! characters read from the
44                                     ! file connected to the default unit
45 CHARACTER(LEN=*),INTENT(IN)        :: set
46                                     ! the set of characters which if found in
47                                     ! the input terminate the read
48 INTEGER,INTENT(IN),OPTIONAL        :: maxlen
49                                     ! if present indicates the maximum
50                                     ! number of characters that will be
51                                     ! read from the file
52 INTEGER,INTENT(OUT),OPTIONAL        :: iostat
53                                     ! if present used to return the status
54                                     ! of the data transfer
55                                     ! if absent errors cause termination
56 ! reads string from the default unit starting at next character in the file and
57 ! terminating at the end of record, occurrence of a character in set,
58 ! or after reading maxlen characters.
59 CHARACTER :: buffer ! characters must be read one at a time to detect
60                ! first terminator character in set
61 INTEGER   :: ist,toread,lenset
62 ist=0
63 lenset = LEN(set)
64 IF(PRESENT(maxlen))THEN
65     toread=maxlen
66 ELSE
67     toread=HUGE(1)
68 ENDIF
69 string = "" ! clears return string
70 DO ! repeatedly read buffer and add to string until EoR
71     ! or maxlen reached

```

```

1      IF(toread <= 0)EXIT
2      READ(*,FMT='(A)',ADVANCE='NO',EOR=9999,IOSTAT=ist) buffer
3      IF( ist /= 0 )THEN
4          IF(PRESENT(iostat)) THEN
5              iostat=ist
6              RETURN
7          ELSE
8              WRITE(*,*) " Error No.",ist, &
9                  " during READ_STRING of varying string on default unit"
10             STOP
11         ENDIF
12     ENDIF
13     ! check for occurrence of set character in buffer
14     DO j = 1,lenset
15         IF(buffer == set(j:j)) GOTO 9999
16     ENDDO
17     string = string//buffer
18     toread = toread - 1
19 ENDDO
20 IF(PRESENT(iostat)) iostat = ist
21 RETURN
22 9999 string = string//buffer
23 IF(PRESENT(iostat)) iostat = ist
24 ENDSUBROUTINE get_d_tset_c

25 SUBROUTINE get_u_tset_c(unit,string,set,maxlen,iostat)
26     INTEGER,INTENT(IN)          :: unit
27                                 ! identifies the input unit which must be
28                                 ! connected for sequential formatted read
29     type(VARYING_STRING),INTENT(OUT) :: string
30                                 ! the string variable to be filled with
31                                 ! characters read from the
32                                 ! file connected to the unit
33     CHARACTER(LEN=*),INTENT(IN)  :: set
34                                 ! the set of characters which if found in
35                                 ! the input terminate the read
36     INTEGER,INTENT(IN),OPTIONAL  :: maxlen
37                                 ! if present indicates the maximum
38                                 ! number of characters that will be
39                                 ! read from the file
40     INTEGER,INTENT(OUT),OPTIONAL :: iostat
41                                 ! if present used to return the status
42                                 ! of the data transfer
43                                 ! if absent errors cause termination
44     ! reads string from unit starting at next character in the file and
45     ! terminating at the end of record, occurrence of a character in set,
46     ! or after reading maxlen characters.
47     CHARACTER :: buffer ! characters must be read one at a time to detect
48                 ! first terminator character in set
49     INTEGER   :: ist,toread,lenset
50     ist=0
51     lenset = LEN(set)
52     IF(PRESENT(maxlen))THEN
53         toread=maxlen
54     ELSE
55         toread=HUGE(1)
56     ENDIF
57     string = "" ! clears return string
58     DO ! repeatedly read buffer and add to string until Eor
59         ! or maxlen reached
60         IF(toread <= 0)EXIT
61         READ(unit,FMT='(A)',ADVANCE='NO',EOR=9999,IOSTAT=ist) buffer
62         IF( ist /= 0 )THEN
63             IF(PRESENT(iostat)) THEN
64                 iostat=ist
65                 RETURN
66             ELSE
67                 WRITE(*,*) " Error No.",ist, &
68                     " during READ_STRING of varying string on unit ",unit
69                 STOP
70             ENDIF
71         ENDIF

```

```

1      ! check for occurrence of set character in buffer
2      DO j = 1,lenset
3          IF(buffer == set(j:j)) GOTO 9999
4      ENDDO
5      string = string//buffer
6      toread = toread - 1
7  ENDDO
8  IF(PRESENT(iostat)) iostat = ist
9  RETURN
10 string = string//buffer
11 IF(PRESENT(iostat)) iostat = ist
12 ENDSUBROUTINE get_u_tset_c

13 !----- Output string procedures -----!
14 SUBROUTINE put_d_s(string,iostat)
15     type(VARYING_STRING),INTENT(IN) :: string
16                                     ! the string variable to be appended to
17                                     ! the current record or to the start of
18                                     ! the next record if there is no
19                                     ! current record
20                                     ! uses the default unit
21     INTEGER,INTENT(OUT),OPTIONAL    :: iostat
22                                     ! if present used to return the status
23                                     ! of the data transfer
24                                     ! if absent errors cause termination
25     INTEGER                          :: ist
26     WRITE(*,FMT='(A)',ADVANCE='NO',IOSTAT=ist) CHAR(string)
27     IF( ist /= 0 )THEN
28         IF(PRESENT(iostat))THEN
29             iostat = ist
30             RETURN
31         ELSE
32             WRITE(*,*) " Error No.",ist, &
33                 " during WRITE_STRING of varying string on default unit"
34         STOP
35     ENDIF
36 ENDIF
37 IF(PRESENT(iostat)) iostat=0
38 ENDSUBROUTINE put_d_s

39 SUBROUTINE put_u_s(unit,string,iostat)
40     INTEGER,INTENT(IN)                :: unit
41                                     ! identifies the output unit which must
42                                     ! be connected for sequential formatted
43                                     ! write
44     type(VARYING_STRING),INTENT(IN) :: string
45                                     ! the string variable to be appended to
46                                     ! the current record or to the start of
47                                     ! the next record if there is no
48                                     ! current record
49     INTEGER,INTENT(OUT),OPTIONAL    :: iostat
50                                     ! if present used to return the status
51                                     ! of the data transfer
52                                     ! if absent errors cause termination
53     WRITE(unit,FMT='(A)',ADVANCE='NO',IOSTAT=ist) CHAR(string)
54     IF( ist /= 0 )THEN
55         IF(PRESENT(iostat))THEN
56             iostat = ist
57             RETURN
58         ELSE
59             WRITE(*,*) " Error No.",ist, &
60                 " during WRITE_STRING of varying string on UNIT ",unit
61         STOP
62     ENDIF
63 ENDIF
64 IF(PRESENT(iostat)) iostat=0
65 ENDSUBROUTINE put_u_s

66 SUBROUTINE put_d_c(string,iostat)
67     CHARACTER(LEN=*),INTENT(IN)      :: string
68                                     ! the character variable to be appended to
69                                     ! the current record or to the start of
70

```

```

1          ! the next record if there is no
2          ! current record
3          ! uses the default unit
4  INTEGER,INTENT(OUT),OPTIONAL  :: iostat
5          ! if present used to return the status
6          ! of the data transfer
7          ! if absent errors cause termination
8
9  INTEGER :: ist
10 WRITE(*,FMT='(A)',ADVANCE='NO',IOSTAT=ist) string
11 IF( ist /= 0 )THEN
12   IF(PRESENT(iostat))THEN
13     iostat = ist
14     RETURN
15   ELSE
16     WRITE(*,*) " Error No.",ist, &
17       " during WRITE_STRING of character on default unit"
18   STOP
19   ENDIF
20 ENDIF
21 IF(PRESENT(iostat)) iostat=0
22 ENDSUBROUTINE put_d_c
23
24 SUBROUTINE put_u_c(unit,string,iostat)
25   INTEGER,INTENT(IN)          :: unit
26   ! identifies the output unit which must
27   ! be connected for sequential formatted
28   ! write
29   CHARACTER(LEN=*),INTENT(IN) :: string
30   ! the character variable to be appended to
31   ! the current record or to the start of
32   ! the next record if there is no
33   ! current record
34   INTEGER,INTENT(OUT),OPTIONAL :: iostat
35   ! if present used to return the status
36   ! of the data transfer
37   ! if absent errors cause termination
38
39   INTEGER :: ist
40   WRITE(unit,FMT='(A)',ADVANCE='NO',IOSTAT=ist) string
41   IF( ist /= 0 )THEN
42     IF(PRESENT(iostat))THEN
43       iostat = ist
44       RETURN
45     ELSE
46       WRITE(*,*) " Error No.",ist," during WRITE_STRING of character on UNIT ",unit
47     STOP
48     ENDIF
49   ENDIF
50 IF(PRESENT(iostat)) iostat=0
51 ENDSUBROUTINE put_u_c
52
53 SUBROUTINE putline_d_s(string,iostat)
54   type(VARYING_STRING),INTENT(IN) :: string
55   ! the string variable to be appended to
56   ! the current record or to the start of
57   ! the next record if there is no
58   ! current record
59   ! uses the default unit
60   INTEGER,INTENT(OUT),OPTIONAL :: iostat
61   ! if present used to return the status
62   ! of the data transfer
63   ! if absent errors cause termination
64
65   ! appends the string to the current record and then ends the record
66   ! leaves the file positioned after the record just completed which then
67   ! becomes the previous and last record in the file.
68   INTEGER          :: ist
69   WRITE(*,FMT='(A,/)',ADVANCE='NO',IOSTAT=ist) CHAR(string)
70   IF( ist /= 0 )THEN
71     IF(PRESENT(iostat))THEN
72       iostat = ist; RETURN
73     ELSE
74       WRITE(*,*) " Error No.",ist, &
75         " during WRITE_LINE of varying string on default unit"

```

```

1      STOP
2      ENDIF
3      ENDIF
4      IF(PRESENT(iostat)) iostat=0
5      ENDSUBROUTINE putline_d_s
6
7      SUBROUTINE putline_u_s(unit,string,iostat)
8      INTEGER,INTENT(IN)      :: unit
9                              ! identifies the output unit which must
10                             ! be connected for sequential formatted
11                             ! write
12      type(VARYING_STRING),INTENT(IN) :: string
13                              ! the string variable to be appended to
14                              ! the current record or to the start of
15                              ! the next record if there is no
16                              ! current record
17      INTEGER,INTENT(OUT),OPTIONAL :: iostat
18                              ! if present used to return the status
19                              ! of the data transfer
20                              ! if absent errors cause termination
21      ! appends the string to the current record and then ends the record
22      ! leaves the file positioned after the record just completed which then
23      ! becomes the previous and last record in the file.
24      INTEGER :: ist
25      WRITE(unit,FMT='(A,/) ',ADVANCE='NO',IOSTAT=ist) CHAR(string)
26      IF( ist /= 0 )THEN
27          IF(PRESENT(iostat))THEN
28              iostat = ist; RETURN
29          ELSE
30              WRITE(*,*) " Error No.",ist, &
31                  " during WRITE_LINE of varying string on UNIT",unit
32          STOP
33          ENDIF
34      ENDIF
35      IF(PRESENT(iostat)) iostat=0
36      ENDSUBROUTINE putline_u_s
37
38      SUBROUTINE putline_d_c(string,iostat)
39      CHARACTER(LEN=*),INTENT(IN)      :: string
40                              ! the character variable to be appended to
41                              ! the current record or to the start of
42                              ! the next record if there is no
43                              ! current record
44                              ! uses the default unit
45      INTEGER,INTENT(OUT),OPTIONAL :: iostat
46                              ! if present used to return the status
47                              ! of the data transfer
48                              ! if absent errors cause termination
49      ! appends the string to the current record and then ends the record
50      ! leaves the file positioned after the record just completed which then
51      ! becomes the previous and last record in the file.
52      INTEGER :: ist
53      WRITE(*,FMT='(A,/) ',ADVANCE='NO',IOSTAT=ist) string
54      IF(PRESENT(iostat))THEN
55          iostat = ist
56          RETURN
57      ELSEIF( ist /= 0 )THEN
58          WRITE(*,*) " Error No.",ist, &
59              " during WRITE_LINE of character on default unit"
60      STOP
61      ENDIF
62      ENDSUBROUTINE putline_d_c
63
64      SUBROUTINE putline_u_c(unit,string,iostat)
65      INTEGER,INTENT(IN)      :: unit
66                              ! identifies the output unit which must
67                              ! be connected for sequential formatted
68                              ! write
69      CHARACTER(LEN=*),INTENT(IN) :: string
70                              ! the character variable to be appended to
71                              ! the current record or to the start of
72                              ! the next record if there is no

```

```

1          ! current record
2  INTEGER, INTENT(OUT), OPTIONAL  :: iostat
3          ! if present used to return the status
4          ! of the data transfer
5          ! if absent errors cause termination
6  ! appends the string to the current record and then ends the record
7  ! leaves the file positioned after the record just completed which then
8  ! becomes the previous and last record in the file.
9  INTEGER :: ist
10 WRITE(unit, FMT='(A,/)', ADVANCE='NO', IOSTAT=ist) string
11 IF(PRESENT(iostat)) THEN
12     iostat = ist
13     RETURN
14 ELSEIF( ist /= 0 ) THEN
15     WRITE(*,*) " Error No.", ist, &
16         " during WRITE_LINE of character on UNIT", unit
17     STOP
18 ENDIF
19 ENDSUBROUTINE putline_u_c
20
21 !----- Insert procedures -----!
22 FUNCTION insert_ss(string, start, substring)
23     type(VARYING_STRING)      :: insert_ss
24     type(VARYING_STRING), INTENT(IN) :: string
25     INTEGER, INTENT(IN)      :: start
26     type(VARYING_STRING), INTENT(IN) :: substring
27     ! calculates result string by inserting the substring into string
28     ! beginning at position start pushing the remainder of the string
29     ! to the right and enlarging it accordingly,
30     ! if start is greater than LEN(string) the substring is simply appended
31     ! to string by concatenation. if start is less than 1
32     ! substring is inserted before string, ie. start is treated as if it were 1
33     CHARACTER, POINTER, DIMENSION(:) :: work
34     INTEGER                          :: ip, is, lsub, ls
35     lsub = LEN(substring); ls = LEN(string)
36     is = MAX(start, 1)
37     ip = MIN(ls+1, is)
38     ALLOCATE(work(1:lsub+ls))
39     work(1:ip-1) = string%chars(1:ip-1)
40     work(ip:ip+lsub-1) = substring%chars
41     work(ip+lsub:lsub+ls) = string%chars(ip:ls)
42     insert_ss%chars => work
43 ENDFUNCTION insert_ss
44
45 FUNCTION insert_sc(string, start, substring)
46     type(VARYING_STRING)      :: insert_sc
47     type(VARYING_STRING), INTENT(IN) :: string
48     INTEGER, INTENT(IN)      :: start
49     CHARACTER(LEN=*), INTENT(IN) :: substring
50     ! calculates result string by inserting the substring into string
51     ! beginning at position start pushing the remainder of the string
52     ! to the right and enlarging it accordingly,
53     ! if start is greater than LEN(string) the substring is simply appended
54     ! to string by concatenation. if start is less than 1
55     ! substring is inserted before string, ie. start is treated as if it were 1
56     CHARACTER, POINTER, DIMENSION(:) :: work
57     INTEGER                          :: ip, is, lsub, ls
58     lsub = LEN(substring); ls = LEN(string)
59     is = MAX(start, 1)
60     ip = MIN(ls+1, is)
61     ALLOCATE(work(1:lsub+ls))
62     work(1:ip-1) = string%chars(1:ip-1)
63     DO i = 1, lsub
64         work(ip-1+i) = substring(i:i)
65     ENDDO
66     work(ip+lsub:lsub+ls) = string%chars(ip:ls)
67     insert_sc%chars => work
68 ENDFUNCTION insert_sc
69
70 FUNCTION insert_cs(string, start, substring)
71     type(VARYING_STRING)      :: insert_cs
72     CHARACTER(LEN=*), INTENT(IN) :: string

```

```

1      INTEGER,INTENT(IN)          :: start
2      type(VARYING_STRING),INTENT(IN) :: substring
3      ! calculates result string by inserting the substring into string
4      ! beginning at position start pushing the remainder of the string
5      ! to the right and enlarging it accordingly,
6      ! if start is greater than LEN(string) the substring is simply appended
7      ! to string by concatenation. if start is less than 1
8      ! substring is inserted before string, ie. start is treated as if it were 1
9      CHARACTER,POINTER,DIMENSION(:) :: work
10     INTEGER                      :: ip,is,lsub,ls
11     lsub = LEN(substring); ls = LEN(string)
12     is = MAX(start,1)
13     ip = MIN(ls+1,is)
14     ALLOCATE(work(1:lsub+ls))
15     DO i=1,ip-1
16         work(i) = string(i:i)
17     ENDDO
18     work(ip:ip+lsub-1) =substring%chars
19     DO i=ip,ls
20         work(i+lsub) = string(i:i)
21     ENDDO
22     insert_cs%chars => work
23 ENDFUNCTION insert_cs
24
25 FUNCTION insert_cc(string,start,substring)
26 type(VARYING_STRING)          :: insert_cc
27 CHARACTER(LEN=*),INTENT(IN) :: string
28 INTEGER,INTENT(IN)          :: start
29 CHARACTER(LEN=*),INTENT(IN) :: substring
30 ! calculates result string by inserting the substring into string
31 ! beginning at position start pushing the remainder of the string
32 ! to the right and enlarging it accordingly,
33 ! if start is greater than LEN(string) the substring is simply appended
34 ! to string by concatenation. if start is less than 1
35 ! substring is inserted before string, ie. start is treated as if it were 1
36 CHARACTER,POINTER,DIMENSION(:) :: work
37 INTEGER                      :: ip,is,lsub,ls
38 lsub = LEN(substring); ls = LEN(string)
39 is = MAX(start,1)
40 ip = MIN(ls+1,is)
41 ALLOCATE(work(1:lsub+ls))
42 DO i=1,ip-1
43     work(i) = string(i:i)
44 ENDDO
45 DO i = 1,lsub
46     work(ip-1+i) = substring(i:i)
47 ENDDO
48 DO i=ip,ls
49     work(i+lsub) = string(i:i)
50 ENDDO
51 insert_cc%chars => work
52 ENDFUNCTION insert_cc
53
54 !----- Replace procedures -----!
55 FUNCTION replace_ss(string,start,substring)
56 type(VARYING_STRING)          :: replace_ss
57 type(VARYING_STRING),INTENT(IN) :: string
58 INTEGER,INTENT(IN)          :: start
59 type(VARYING_STRING),INTENT(IN) :: substring
60 ! calculates the result string by the following actions:
61 ! inserts the substring into string beginning at position
62 ! start replacing the following LEN(substring) characters of the string
63 ! and enlarging string if necessary. if start is greater than LEN(string)
64 ! substring is simply appended to string by concatenation. If start is less
65 ! than 1, substring replaces characters in string starting at 1
66 CHARACTER,POINTER,DIMENSION(:) :: work
67 INTEGER                      :: ip,is,nw,lsub,ls
68 lsub = LEN(substring); ls = LEN(string)
69 is = MAX(start,1)
70 ip = MIN(ls+1,is)
71 nw = MAX(ls,ip+lsub-1)
72 ALLOCATE(work(1:nw))

```



```

1      work(1:ip-1) = string%chars(1:ip-1)
2      work(ip:ip+lsub-1) = substring%chars
3      work(ip+lsub:nw) = string%chars(ip+lsub:ls)
4      replace_ss%chars => work
5      ENDFUNCTION replace_ss
6
7      FUNCTION replace_ss_sf(string,start,finish,substring)
8      type(VARYING_STRING)      :: replace_ss_sf
9      type(VARYING_STRING),INTENT(IN) :: string
10     INTEGER,INTENT(IN)      :: start,finish
11     type(VARYING_STRING),INTENT(IN) :: substring
12     ! calculates the result string by the following actions:
13     ! inserts the substring into string beginning at position
14     ! start replacing the following finish-start+1 characters of the string
15     ! and enlarging or shrinking the string if necessary.
16     ! If start is greater than LEN(string) substring is simply appended to string
17     ! by concatenation. If start is less than 1, start = 1 is used
18     ! If finish is greater than LEN(string), finish = LEN(string) is used
19     ! If finish is less than start, substring is inserted before start
20     CHARACTER,POINTER,DIMENSION(:) :: work
21     INTEGER      :: ip,is,if,nw,lsub,ls
22     lsub = LEN(substring); ls = LEN(string)
23     is = MAX(start,1)
24     ip = MIN(ls+1,is)
25     if = MAX(ip-1,MIN(finish,ls))
26     nw = lsub + ls - if+ip-1
27     ALLOCATE(work(1:nw))
28     work(1:ip-1) = string%chars(1:ip-1)
29     work(ip:ip+lsub-1) = substring%chars
30     work(ip+lsub:nw) = string%chars(if+1:ls)
31     replace_ss_sf%chars => work
32     ENDFUNCTION replace_ss_sf
33
34     FUNCTION replace_sc(string,start,substring)
35     type(VARYING_STRING)      :: replace_sc
36     type(VARYING_STRING),INTENT(IN) :: string
37     INTEGER,INTENT(IN)      :: start
38     CHARACTER(LEN=*),INTENT(IN) :: substring
39     ! calculates the result string by the following actions:
40     ! inserts the characters from substring into string beginning at position
41     ! start replacing the following LEN(substring) characters of the string
42     ! and enlarging string if necessary. If start is greater than LEN(string)
43     ! substring is simply appended to string by concatenation. If start is less
44     ! than 1, substring replaces characters in string starting at 1
45     CHARACTER,POINTER,DIMENSION(:) :: work
46     INTEGER      :: ip,is,nw,lsub,ls
47     lsub = LEN(substring); ls = LEN(string)
48     is = MAX(start,1)
49     ip = MIN(ls+1,is)
50     nw = MAX(ls,ip+lsub-1)
51     ALLOCATE(work(1:nw))
52     work(1:ip-1) = string%chars(1:ip-1)
53     DO i = 1,lsub
54         work(ip-1+i) = substring(i:i)
55     ENDDO
56     work(ip+lsub:nw) = string%chars(ip+lsub:ls)
57     replace_sc%chars => work
58     ENDFUNCTION replace_sc
59
60     FUNCTION replace_sc_sf(string,start,finish,substring)
61     type(VARYING_STRING)      :: replace_sc_sf
62     type(VARYING_STRING),INTENT(IN) :: string
63     INTEGER,INTENT(IN)      :: start,finish
64     CHARACTER(LEN=*),INTENT(IN) :: substring
65     ! calculates the result string by the following actions:
66     ! inserts the substring into string beginning at position
67     ! start replacing the following finish-start+1 characters of the string
68     ! and enlarging or shrinking the string if necessary.
69     ! If start is greater than LEN(string) substring is simply appended to string
70     ! by concatenation. If start is less than 1, start = 1 is used
71     ! If finish is greater than LEN(string), finish = LEN(string) is used
72     ! If finish is less than start, substring is inserted before start

```

```

1  CHARACTER, POINTER, DIMENSION(:) :: work
2  INTEGER                          :: ip, is, if, nw, lsub, ls
3  lsub = LEN(substring); ls = LEN(string)
4  is = MAX(start, 1)
5  ip = MIN(ls+1, is)
6  if = MAX(ip-1, MIN(finish, ls))
7  nw = lsub + ls - if+ip-1
8  ALLOCATE(work(1:nw))
9  work(1:ip-1) = string%chars(1:ip-1)
10 DO i = 1, lsub
11     work(ip-1+i) = substring(i:i)
12 ENDDO
13 work(ip+lsub:nw) = string%chars(if+1:ls)
14 replace_sc_sf%chars => work
15 ENDFUNCTION replace_sc_sf

16 FUNCTION replace_cs(string, start, substring)
17 type(VARYING_STRING)      :: replace_cs
18 CHARACTER(LEN=*) , INTENT(IN)  :: string
19 INTEGER, INTENT(IN)        :: start
20 type(VARYING_STRING) , INTENT(IN) :: substring
21 ! calculates the result string by the following actions:
22 ! inserts the substring into string beginning at position
23 ! start replacing the following LEN(substring) characters of the string
24 ! and enlarging string if necessary. if start is greater than LEN(string)
25 ! substring is simply appended to string by concatenation. If start is less
26 ! than 1, substring replaces characters in string starting at 1
27 CHARACTER, POINTER, DIMENSION(:) :: work
28 INTEGER                          :: ip, is, nw, lsub, ls
29 lsub = LEN(substring); ls = LEN(string)
30 is = MAX(start, 1)
31 ip = MIN(ls+1, is)
32 nw = MAX(ls, ip+lsub-1)
33 ALLOCATE(work(1:nw))
34 DO i=1, ip-1
35     work(i) = string(i:i)
36 ENDDO
37 work(ip:ip+lsub-1) = substring%chars
38 DO i=ip+lsub, nw
39     work(i) = string(i:i)
40 ENDDO
41 replace_cs%chars => work
42 ENDFUNCTION replace_cs

43
44 FUNCTION replace_cs_sf(string, start, finish, substring)
45 type(VARYING_STRING)      :: replace_cs_sf
46 CHARACTER(LEN=*) , INTENT(IN)  :: string
47 INTEGER, INTENT(IN)        :: start, finish
48 type(VARYING_STRING) , INTENT(IN) :: substring
49 ! calculates the result string by the following actions:
50 ! inserts the substring into string beginning at position
51 ! start replacing the following finish-start+1 characters of the string
52 ! and enlarging or shrinking the string if necessary.
53 ! If start is greater than LEN(string) substring is simply appended to string
54 ! by concatenation. If start is less than 1, start = 1 is used
55 ! If finish is greater than LEN(string), finish = LEN(string) is used
56 ! If finish is less than start, substring is inserted before start
57 CHARACTER, POINTER, DIMENSION(:) :: work
58 INTEGER                          :: ip, is, if, nw, lsub, ls
59 lsub = LEN(substring); ls = LEN(string)
60 is = MAX(start, 1)
61 ip = MIN(ls+1, is)
62 if = MAX(ip-1, MIN(finish, ls))
63 nw = lsub + ls - if+ip-1
64 ALLOCATE(work(1:nw))
65 DO i=1, ip-1
66     work(i) = string(i:i)
67 ENDDO
68 work(ip:ip+lsub-1) = substring%chars
69 DO i=1, nw-ip-lsub+1
70     work(i+ip+lsub-1) = string(if+i:if+i)
71 ENDDO

```

```

1  replace_cs_sf%chars => work
2  ENDFUNCTION replace_cs_sf

3  FUNCTION replace_cc(string,start,substring)
4  type(VARYING_STRING)      :: replace_cc
5  CHARACTER(LEN=*) ,INTENT(IN)  :: string
6  INTEGER ,INTENT(IN)      :: start
7  CHARACTER(LEN=*) ,INTENT(IN)  :: substring
8  ! calculates the result string by the following actions:
9  ! inserts the characters from substring into string beginning at position
10 ! start replacing the following LEN(substring) characters of the string
11 ! and enlarging string if necessary. If start is greater than LEN(string)
12 ! substring is simply appended to string by concatenation. If start is less
13 ! than 1, substring replaces characters in string starting at 1
14 CHARACTER,POINTER,DIMENSION(:) :: work
15 INTEGER      :: ip,is,nw,ls,ls
16 ls = LEN(substring); ls = LEN(string)
17 is = MAX(start,1)
18 ip = MIN(ls+1,is)
19 nw = MAX(ls,ip+ls-1)
20 ALLOCATE(work(1:nw))
21 DO i=1,ip-1
22   work(i) = string(i:i)
23 ENDDO
24 DO i=1,ls
25   work(ip-1+i) = substring(i:i)
26 ENDDO
27 DO i=ip+ls,nw
28   work(i) = string(i:i)
29 ENDDO
30 replace_cc%chars => work
31 ENDFUNCTION replace_cc

32
33 FUNCTION replace_cc_sf(string,start,finish,substring)
34 type(VARYING_STRING)      :: replace_cc_sf
35 CHARACTER(LEN=*) ,INTENT(IN)  :: string
36 INTEGER ,INTENT(IN)      :: start,finish
37 CHARACTER(LEN=*) ,INTENT(IN)  :: substring
38 ! calculates the result string by the following actions:
39 ! inserts the substring into string beginning at position
40 ! start replacing the following finish-start+1 characters of the string
41 ! and enlarging or shrinking the string if necessary.
42 ! If start is greater than LEN(string) substring is simply appended to string
43 ! by concatenation. If start is less than 1, start = 1 is used
44 ! If finish is greater than LEN(string), finish = LEN(string) is used
45 ! If finish is less than start, substring is inserted before start
46 CHARACTER,POINTER,DIMENSION(:) :: work
47 INTEGER      :: ip,is,if,nw,ls,ls
48 ls = LEN(substring); ls = LEN(string)
49 is = MAX(start,1)
50 ip = MIN(ls+1,is)
51 if = MAX(ip-1,MIN(finish,ls))
52 nw = ls + ls - if+ip-1
53 ALLOCATE(work(1:nw))
54 DO i=1,ip-1
55   work(i) = string(i:i)
56 ENDDO
57 DO i=1,ls
58   work(i+ip-1) = substring(i:i)
59 ENDDO
60 DO i=1,nw-ip-ls+1
61   work(i+ip+ls-1) = string(if+i:if+i)
62 ENDDO
63 replace_cc_sf%chars => work
64 ENDFUNCTION replace_cc_sf

65
66 FUNCTION replace_sss(string,target,substring,every,back)
67 type(VARYING_STRING)      :: replace_sss
68 type(VARYING_STRING),INTENT(IN) :: string,target,substring
69 LOGICAL ,INTENT(IN),OPTIONAL  :: every,back
70 ! calculates the result string by the following actions:
! searches for occurrences of target in string, and replaces these with

```

```

1      ! substring. if back present with value true search is backward otherwise
2      ! search is done forward. if every present with value true all occurrences
3      ! of target in string are replaced, otherwise only the first found is
4      ! replaced. if target is not found the result is the same as string.
5      LOGICAL                :: dir_switch, rep_search
6      CHARACTER,DIMENSION(:),POINTER :: work,temp
7      INTEGER                :: ls,lt,lsub,ipos,ipow
8      ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
9      IF(lt==0)THEN
10         WRITE(*,*) " Zero length target in REPLACE"
11         STOP
12     ENDIF
13     ALLOCATE(work(1:ls)); work = string%chars
14     IF( PRESENT(back) )THEN
15         dir_switch = back
16     ELSE
17         dir_switch = .FALSE.
18     ENDIF
19     IF( PRESENT(every) )THEN
20         rep_search = every
21     ELSE
22         rep_search = .FALSE.
23     ENDIF
24     IF( dir_switch )THEN ! backwards search
25         ipos = ls-lt+1
26         DO
27             IF( ipos < 1 )EXIT ! search past start of string
28             ! test for occurrence of target in string at this position
29             IF( ALL(string%chars(ipos:ipos+lt-1) == target%chars) )THEN
30                 ! match found allocate space for string with this occurrence of
31                 ! target replaced by substring
32                 ALLOCATE(temp(1:SIZE(work)+lsub-lt))
33                 ! copy work into temp replacing this occurrence of target by
34                 ! substring
35                 temp(1:ipos-1) = work(1:ipos-1)
36                 temp(ipos:ipos+lsub-1) = substring%chars
37                 temp(ipos+lsub:) = work(ipos+lt:)
38                 work => temp ! make new version of work point at the temp space
39                 IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
40                 ! move search and replacement positions over the effected positions
41                 ipos = ipos-lt+1
42             ENDIF
43             ipos=ipos-1
44         ENDDO
45     ELSE ! forward search
46         ipos = 1; ipow = 1
47         DO
48             IF( ipos > ls-lt+1 )EXIT ! search past end of string
49             ! test for occurrence of target in string at this position
50             IF( ALL(string%chars(ipos:ipos+lt-1) == target%chars) )THEN
51                 ! match found allocate space for string with this occurrence of
52                 ! target replaced by substring
53                 ALLOCATE(temp(1:SIZE(work)+lsub-lt))
54                 ! copy work into temp replacing this occurrence of target by
55                 ! substring
56                 temp(1:ipow-1) = work(1:ipow-1)
57                 temp(ipow:ipow+lsub-1) = substring%chars
58                 temp(ipow+lsub:) = work(ipow+lt:)
59                 work => temp ! make new version of work point at the temp space
60                 IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
61                 ! move search and replacement positions over the effected positions
62                 ipos = ipos+lt-1; ipow = ipow+lsub-1
63             ENDIF
64             ipos=ipos+1; ipow=ipow+1
65         ENDDO
66     ENDIF
67     replace_sss%chars => work
68 ENDFUNCTION replace_sss

69 FUNCTION replace_ssc(string,target,substring,every,back)
70     type(VARYING_STRING)                :: replace_ssc
71     type(VARYING_STRING),INTENT(IN) :: string,target

```

```

1 CHARACTER(LEN=*),INTENT(IN)      :: substring
2 LOGICAL,INTENT(IN),OPTIONAL      :: every,back
3 ! calculates the result string by the following actions:
4 ! searches for occurrences of target in string, and replaces these with
5 ! substring. if back present with value true search is backward otherwise
6 ! search is done forward. if every present with value true all occurrences
7 ! of target in string are replaced, otherwise only the first found is
8 ! replaced. if target is not found the result is the same as string.
9 LOGICAL                          :: dir_switch, rep_search
10 CHARACTER,DIMENSION(:),POINTER  :: work,temp
11 INTEGER                          :: ls,lt,lsub,ipos,ipow
12 ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
13 IF(lt==0)THEN
14   WRITE(*,*) " Zero length target in REPLACE"
15   STOP
16 ENDIF
17 ALLOCATE(work(1:ls)); work = string%chars
18 IF( PRESENT(back) )THEN
19   dir_switch = back
20 ELSE
21   dir_switch = .FALSE.
22 ENDIF
23 IF( PRESENT(every) )THEN
24   rep_search = every
25 ELSE
26   rep_search = .FALSE.
27 ENDIF
28 IF( dir_switch )THEN ! backwards search
29   ipos = ls-lt+1
30   DO
31     IF( ipos < 1 )EXIT ! search past start of string
32     ! test for occurrence of target in string at this position
33     IF( ALL(string%chars(ipos:ipos+lt-1) == target%chars) )THEN
34       ! match found allocate space for string with this occurrence of
35       ! target replaced by substring
36       ALLOCATE(temp(1:SIZE(work)+lsub-lt))
37       ! copy work into temp replacing this occurrence of target by
38       ! substring
39       temp(1:ipos-1) = work(1:ipos-1)
40       DO i=1,lsub
41         temp(i+ipos-1) = substring(i:i)
42       ENDDO
43       temp(ipos+lsub:) = work(ipos+lt:)
44       work => temp ! make new version of work point at the temp space
45       IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
46       ! move search and replacement positions over the effected positions
47       ipos = ipos-lt+1
48     ENDIF
49     ipos=ipos-1
50   ENDDO
51 ELSE ! forward search
52   ipos = 1; ipow = 1
53   DO
54     IF( ipos > ls-lt+1 )EXIT ! search past end of string
55     ! test for occurrence of target in string at this position
56     IF( ALL(string%chars(ipos:ipos+lt-1) == target%chars) )THEN
57       ! match found allocate space for string with this occurrence of
58       ! target replaced by substring
59       ALLOCATE(temp(1:SIZE(work)+lsub-lt))
60       ! copy work into temp replacing this occurrence of target by
61       ! substring
62       temp(1:ipow-1) = work(1:ipow-1)
63       DO i=1,lsub
64         temp(i+ipow-1) = substring(i:i)
65       ENDDO
66       temp(ipow+lsub:) = work(ipow+lt:)
67       work => temp ! make new version of work point at the temp space
68       IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
69       ! move search and replacement positions over the effected positions
70       ipos = ipos+lt-1; ipow = ipow+lsub-1
71     ENDIF
72     ipos=ipos+1; ipow=ipow+1

```

```

1      ENDDO
2      ENDIF
3      replace_ssc%chars => work
4  ENDFUNCTION replace_ssc

5  FUNCTION replace_scs(string,target,substring,every,back)
6      type(VARYING_STRING)          :: replace_scs
7      type(VARYING_STRING),INTENT(IN) :: string,substring
8      CHARACTER(LEN=*),INTENT(IN)   :: target
9      LOGICAL,INTENT(IN),OPTIONAL   :: every,back
10     ! calculates the result string by the following actions:
11     ! searches for occurrences of target in string, and replaces these with
12     ! substring. if back present with value true search is backward otherwise
13     ! search is done forward. if every present with value true all occurrences
14     ! of target in string are replaced, otherwise only the first found is
15     ! replaced. if target is not found the result is the same as string.
16     LOGICAL          :: dir_switch, rep_search
17     CHARACTER,DIMENSION(:),POINTER :: work,temp,tget
18     INTEGER          :: ls,lt,lsub,ipos,ipow
19     ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
20     IF(lt==0)THEN
21         WRITE(*,*) " Zero length target in REPLACE"
22         STOP
23     ENDIF
24     ALLOCATE(work(1:ls)); work = string%chars
25     ALLOCATE(tget(1:lt))
26     DO i=1,lt
27         tget(i) = target(i:i)
28     ENDDO
29     IF( PRESENT(back) )THEN
30         dir_switch = back
31     ELSE
32         dir_switch = .FALSE.
33     ENDIF
34     IF( PRESENT(every) )THEN
35         rep_search = every
36     ELSE
37         rep_search = .FALSE.
38     ENDIF
39     IF( dir_switch )THEN ! backwards search
40         ipos = ls-lt+1
41         DO
42             IF( ipos < 1 )EXIT ! search past start of string
43             ! test for occurrence of target in string at this position
44             IF( ALL(string%chars(ipos:ipos+lt-1) == tget) )THEN
45                 ! match found allocate space for string with this occurrence of
46                 ! target replaced by substring
47                 ALLOCATE(temp(1:SIZE(work)+lsub-lt))
48                 ! copy work into temp replacing this occurrence of target by
49                 ! substring
50                 temp(1:ipos-1) = work(1:ipos-1)
51                 temp(ipos:ipos+lsub-1) = substring%chars
52                 temp(ipos+lsub:) = work(ipos+lt:)
53                 work => temp ! make new version of work point at the temp space
54                 IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
55                 ! move search and replacement positions over the effected positions
56                 ipos = ipos-lt+1
57             ENDIF
58             ipos=ipos-1
59         ENDDO
60     ELSE ! forward search
61         ipos = 1; ipow = 1
62         DO
63             IF( ipos > ls-lt+1 )EXIT ! search past end of string
64             ! test for occurrence of target in string at this position
65             IF( ALL(string%chars(ipos:ipos+lt-1) == tget) )THEN
66                 ! match found allocate space for string with this occurrence of
67                 ! target replaced by substring
68                 ALLOCATE(temp(1:SIZE(work)+lsub-lt))
69                 ! copy work into temp replacing this occurrence of target by
70                 ! substring
71                 temp(1:ipow-1) = work(1:ipow-1)

```

```

1      temp(ipow:ipow+lsub-1) = substring%chars
2      temp(ipow+lsub:) = work(ipow+lt:)
3      work => temp ! make new version of work point at the temp space
4      IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
5      ! move search and replacement positions over the effected positions
6      ipos = ipos+lt-1; ipow = ipow+lsub-1
7      ENDIF
8      ipos=ipos+1; ipow=ipow+1
9      ENDDO
10     ENDIF
11     replace_scs%chars => work
12 ENDFUNCTION replace_scs

13 FUNCTION replace_scc(string,target,substring,every,back)
14 type(VARYING_STRING)      :: replace_scc
15 type(VARYING_STRING),INTENT(IN) :: string
16 CHARACTER(LEN=*),INTENT(IN)  :: target,substring
17 LOGICAL,INTENT(IN),OPTIONAL  :: every,back
18 ! calculates the result string by the following actions:
19 ! searches for occurences of target in string, and replaces these with
20 ! substring. if back present with value true search is backward otherwise
21 ! search is done forward. if every present with value true all accurences
22 ! of target in string are replaced, otherwise only the first found is
23 ! replaced. if target is not found the result is the same as string.
24 LOGICAL      :: dir_switch, rep_search
25 CHARACTER,DIMENSION(:),POINTER :: work,temp,tget
26 INTEGER      :: ls,lt,lsub,ipos,ipow
27 ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
28 IF(lt==0)THEN
29     WRITE(*,*) " Zero length target in REPLACE"
30     STOP
31 ENDIF
32 ALLOCATE(work(1:ls)); work = string%chars
33 ALLOCATE(tget(1:lt))
34 DO i=1,lt
35     tget(i) = target(i:i)
36 ENDDO
37 IF( PRESENT(back) )THEN
38     dir_switch = back
39 ELSE
40     dir_switch = .FALSE.
41 ENDIF
42 IF( PRESENT(every) )THEN
43     rep_search = every
44 ELSE
45     rep_search = .FALSE.
46 ENDIF
47 IF( dir_switch )THEN ! backwards search
48     ipos = ls-lt+1
49     DO
50         IF( ipos < 1 )EXIT ! search past start of string
51         ! test for occurance of target in string at this position
52         IF( ALL(string%chars(ipos:ipos+lt-1) == tget) )THEN
53             ! match found allocate space for string with this occurrence of
54             ! target replaced by substring
55             ALLOCATE(temp(1:SIZE(work)+lsub-lt))
56             ! copy work into temp replacing this occurrence of target by
57             ! substring
58             temp(1:ipos-1) = work(1:ipos-1)
59             DO i=1,lsub
60                 temp(i+ipos-1) = substring(i:i)
61             ENDDO
62             temp(ipos+lsub:) = work(ipos+lt:)
63             work => temp ! make new version of work point at the temp space
64             IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
65             ! move search and replacement positions over the effected positions
66             ipos = ipos-lt+1
67         ENDIF
68         ipos=ipos-1
69     ENDDO
70 ELSE ! forward search
71     ipos = 1; ipow = 1

```

```

1      DO
2          IF( ipos > ls-lt+1 )EXIT ! search past end of string
3          ! test for occurrence of target in string at this position
4          IF( ALL(string%chars(ipos:ipos+lt-1) == tget) )THEN
5              ! match found allocate space for string with this occurrence of
6              ! target replaced by substring
7              ALLOCATE(temp(1:SIZE(work)+lsub-lt))
8              ! copy work into temp replacing this occurrence of target by
9              ! substring
10             temp(1:ipow-1) = work(1:ipow-1)
11             DO i=1,lsub
12                 temp(i+ipow-1) = substring(i:i)
13             ENDDO
14             temp(ipow+lsub:) = work(ipow+lt:)
15             work => temp ! make new version of work point at the temp space
16             IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
17             ! move search and replacement positions over the effected positions
18             ipos = ipos+lt-1; ipow = ipow+lsub-1
19         ENDIF
20         ipos=ipos+1; ipow=ipow+1
21     ENDDO
22 ENDIF
23 replace_scc%chars => work
24 ENDFUNCTION replace_scc

25 FUNCTION replace_css(string,target,substring,every,back)
26     type(VARYING_STRING)          :: replace_css
27     CHARACTER(LEN=*),INTENT(IN)   :: string
28     type(VARYING_STRING),INTENT(IN) :: target,substring
29     LOGICAL,INTENT(IN),OPTIONAL   :: every,back
30     ! calculates the result string by the following actions:
31     ! searches for occurrences of target in string, and replaces these with
32     ! substring. if back present with value true search is backward otherwise
33     ! search is done forward. if every present with value true all occurrences
34     ! of target in string are replaced, otherwise only the first found is
35     ! replaced. if target is not found the result is the same as string.
36     LOGICAL                        :: dir_switch, rep_search
37     CHARACTER,DIMENSION(:),POINTER :: work,temp,str
38     INTEGER                        :: ls,lt,lsub,ipos,ipow
39     ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
40     IF(lt==0)THEN
41         WRITE(*,*) " Zero length target in REPLACE"
42         STOP
43     ENDIF
44     ALLOCATE(work(1:ls)); ALLOCATE(str(1:ls))
45     DO i=1,ls
46         str(i) = string(i:i)
47     ENDDO
48     work = str
49     IF( PRESENT(back) )THEN
50         dir_switch = back
51     ELSE
52         dir_switch = .FALSE.
53     ENDIF
54     IF( PRESENT(every) )THEN
55         rep_search = every
56     ELSE
57         rep_search = .FALSE.
58     ENDIF
59     IF( dir_switch )THEN ! backwards search
60         ipos = ls-lt+1
61         DO
62             IF( ipos < 1 )EXIT ! search past start of string
63             ! test for occurrence of target in string at this position
64             IF( ALL(str(ipos:ipos+lt-1) == target%chars) )THEN
65                 ! match found allocate space for string with this occurrence of
66                 ! target replaced by substring
67                 ALLOCATE(temp(1:SIZE(work)+lsub-lt))
68                 ! copy work into temp replacing this occurrence of target by
69                 ! substring
70                 temp(1:ipos-1) = work(1:ipos-1)
71                 temp(ipos:ipos+lsub-1) = substring%chars

```



```

1      temp(ipos+lsub:) = work(ipos+lt:)
2      work => temp ! make new version of work point at the temp space
3      IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
4      ! move search and replacement positions over the effected positions
5      ipos = ipos-lt+1
6      ENDDIF
7      ipos=ipos-1
8      ENDDO
9      ELSE ! forward search
10     ipos = 1; ipow = 1
11     DO
12     IF( ipos > ls-lt+1 )EXIT ! search past end of string
13     ! test for occurance of target in string at this position
14     IF( ALL(str(ipos:ipos+lt-1) == target%chars) )THEN
15     ! match found allocate space for string with this occurrence of
16     ! target replaced by substring
17     ALLOCATE(temp(1:SIZE(work)+lsub-lt))
18     ! copy work into temp replacing this occurrence of target by
19     ! substring
20     temp(1:ipow-1) = work(1:ipow-1)
21     temp(ipow:ipow+lsub-1) = substring%chars
22     temp(ipow+lsub:) = work(ipow+lt:)
23     work => temp ! make new version of work point at the temp space
24     IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
25     ! move search and replacement positions over the effected positions
26     ipos = ipos+lt-1; ipow = ipow+lsub-1
27     ENDDIF
28     ipos=ipos+1; ipow=ipow+1
29     ENDDO
30     ENDDIF
31     replace_css%chars => work
32 ENDFUNCTION replace_css

33 FUNCTION replace_csc(string,target,substring,every,back)
34 type(VARYING_STRING)      :: replace_csc
35 type(VARYING_STRING),INTENT(IN) :: target
36 CHARACTER(LEN=*),INTENT(IN)  :: string,substring
37 LOGICAL,INTENT(IN),OPTIONAL  :: every,back
38 ! calculates the result string by the following actions:
39 ! searches for occurences of target in string, and replaces these with
40 ! substring. if back present with value true search is backward otherwise
41 ! search is done forward. if every present with value true all accurences
42 ! of target in string are replaced, otherwise only the first found is
43 ! replaced. if target is not found the result is the same as string.
44 LOGICAL      :: dir_switch, rep_search
45 CHARACTER,DIMENSION(:),POINTER :: work,temp,str
46 INTEGER      :: ls,lt,lsub,ipos,ipow
47 ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
48 IF(lt==0)THEN
49     WRITE(*,*) " Zero length target in REPLACE"
50     STOP
51 ENDDIF
52 ALLOCATE(work(1:ls)); ALLOCATE(str(1:ls))
53 DO i=1,ls
54     str(i) = string(i:i)
55 ENDDO
56 work = str
57 IF( PRESENT(back) )THEN
58     dir_switch = back
59 ELSE
60     dir_switch = .FALSE.
61 ENDDIF
62 IF( PRESENT(every) )THEN
63     rep_search = every
64 ELSE
65     rep_search = .FALSE.
66 ENDDIF
67 IF( dir_switch )THEN ! backwards search
68     ipos = ls-lt+1
69     DO
70     IF( ipos < 1 )EXIT ! search past start of string
71     ! test for occurance of target in string at this position

```

```

1      IF( ALL(str(ipos:ipow+lt-1) == target%chars) )THEN
2          ! match found allocate space for string with this occurrence of
3          ! target replaced by substring
4          ALLOCATE(temp(1:SIZE(work)+lsub-lt))
5          ! copy work into temp replacing this occurrence of target by
6          ! substring
7          temp(1:ipow-1) = work(1:ipow-1)
8          DO i=1,lsub
9              temp(i+ipow-1) = substring(i:i)
10             ENDDO
11             temp(ipow+lsub:) = work(ipow+lt:)
12             work => temp ! make new version of work point at the temp space
13             IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
14             ! move search and replacement positions over the effected positions
15             ipos = ipow+lt+1
16         ENDIF
17         ipos=ipow-1
18     ENDDO
19 ELSE ! forward search
20     ipos = 1; ipow = 1
21     DO
22         IF( ipos > ls-lt+1 )EXIT ! search past end of string
23         ! test for occurrence of target in string at this position
24         IF( ALL(str(ipos:ipow+lt-1) == target%chars) )THEN
25             ! match found allocate space for string with this occurrence of
26             ! target replaced by substring
27             ALLOCATE(temp(1:SIZE(work)+lsub-lt))
28             !copy work into temp replacing this occurrence of target by
29             ! substring
30             temp(1:ipow-1) = work(1:ipow-1)
31             DO i=1,lsub
32                 temp(i+ipow-1) = substring(i:i)
33             ENDDO
34             temp(ipow+lsub:) = work(ipow+lt:)
35             work => temp ! make new version of work point at the temp space
36             IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
37             ! move search and replacement positions over the effected positions
38             ipos = ipow+lt-1; ipow = ipow+lsub-1
39         ENDIF
40         ipos=ipos+1; ipow=ipow+1
41     ENDDO
42     ENDIF
43     replace_csc%chars => work
44 ENDFUNCTION replace_csc

45 FUNCTION replace_ccs(string,target,substring,every,back)
46     type(VARYING_STRING)          :: replace_ccs
47     type(VARYING_STRING),INTENT(IN) :: substring
48     CHARACTER(LEN=*),INTENT(IN)    :: string,target
49     LOGICAL,INTENT(IN),OPTIONAL    :: every,back
50     ! calculates the result string by the following actions:
51     ! searches for occurrences of target in string, and replaces these with
52     ! substring. if back present with value true search is backward otherwise
53     ! search is done forward. if every present with value true all occurrences
54     ! of target in string are replaced, otherwise only the first found is
55     ! replaced. if target is not found the result is the same as string.
56     LOGICAL                        :: dir_switch, rep_search
57     CHARACTER,DIMENSION(:),POINTER :: work,temp
58     INTEGER                        :: ls,lt,lsub,ipos,ipow
59     ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
60     IF(lt==0)THEN
61         WRITE(*,*) " Zero length target in REPLACE"
62         STOP
63     ENDIF
64     ALLOCATE(work(1:ls))
65     DO i=1,ls
66         work(i) = string(i:i)
67     ENDDO
68     IF( PRESENT(back) )THEN
69         dir_switch = back
70     ELSE
71         dir_switch = .FALSE.

```

```

1  ENDIF
2  IF( PRESENT(every) )THEN
3    rep_search = every
4  ELSE
5    rep_search = .FALSE.
6  ENDIF
7  IF( dir_switch )THEN ! backwards search
8    ipos = ls-lt+1
9    DO
10     IF( ipos < 1 )EXIT ! search past start of string
11     ! test for occurrence of target in string at this position
12     IF( string(ipos:ipos+lt-1) == target )THEN
13       ! match found allocate space for string with this occurrence of
14       ! target replaced by substring
15       ALLOCATE(temp(1:SIZE(work)+lsub-lt))
16       ! copy work into temp replacing this occurrence of target by
17       ! substring
18       temp(1:ipos-1) = work(1:ipos-1)
19       temp(ipos:ipos+lsub-1) = substring%chars
20       temp(ipos+lsub:) = work(ipos+lt:)
21       work => temp ! make new version of work point at the temp space
22       IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
23       ! move search and replacement positions over the effected positions
24       ipos = ipos-lt+1
25     ENDIF
26     ipos=ipos-1
27   ENDDO
28 ELSE ! forward search
29   ipos = 1; ipow = 1
30   DO
31     IF( ipos > ls-lt+1 )EXIT ! search past end of string
32     ! test for occurrence of target in string at this position
33     IF( string(ipos:ipos+lt-1) == target )THEN
34       ! match found allocate space for string with this occurrence of
35       ! target replaced by substring
36       ALLOCATE(temp(1:SIZE(work)+lsub-lt))
37       ! copy work into temp replacing this occurrence of target by
38       ! substring
39       temp(1:ipow-1) = work(1:ipow-1)
40       temp(ipow:ipow+lsub-1) = substring%chars
41       temp(ipow+lsub:) = work(ipow+lt:)
42       work => temp ! make new version of work point at the temp space
43       IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
44       ! move search and replacement positions over the effected positions
45       ipos = ipos+lt-1; ipow = ipow+lsub-1
46     ENDIF
47     ipos=ipos+1; ipow=ipow+1
48   ENDDO
49 ENDIF
50 replace_ccs%chars => work
51 ENDFUNCTION replace_ccs

52 FUNCTION replace_ccc(string,target,substring,every,back)
53 type(VARYING_STRING)      :: replace_ccc
54 CHARACTER(LEN=*) ,INTENT(IN)  :: string,target,substring
55 LOGICAL ,INTENT(IN),OPTIONAL  :: every,back
56 ! calculates the result string by the following actions:
57 ! searches for occurrences of target in string, and replaces these with
58 ! substring. if back present with value true search is backward otherwise
59 ! search is done forward. if every present with value true all occurrences
60 ! of target in string are replaced, otherwise only the first found is
61 ! replaced. if target is not found the result is the same as string.
62 LOGICAL                      :: dir_switch, rep_search
63 CHARACTER,DIMENSION(:),POINTER :: work,temp
64 INTEGER                      :: ls,lt,lsub,ipos,ipow
65 ls = LEN(string); lt = LEN(target); lsub = LEN(substring)
66 IF(lt==0)THEN
67   WRITE(*,*) " Zero length target in REPLACE"
68   STOP
69 ENDIF
70 ALLOCATE(work(1:ls))
71 DO i=1,ls

```

```

1      work(i) = string(i:i)
2      ENDDO
3      IF( PRESENT(back) )THEN
4          dir_switch = back
5      ELSE
6          dir_switch = .FALSE.
7      ENDIF
8      IF( PRESENT(every) )THEN
9          rep_search = every
10     ELSE
11         rep_search = .FALSE.
12     ENDIF
13     IF( dir_switch )THEN ! backwards search
14         ipos = ls-lt+1
15         DO
16             IF( ipos < 1 )EXIT ! search past start of string
17             ! test for occurrence of target in string at this position
18             IF( string(ipos:ipos+lt-1) == target )THEN
19                 ! match found allocate space for string with this occurrence of
20                 ! target replaced by substring
21                 ALLOCATE(temp(1:SIZE(work)+lsub-lt))
22                 ! copy work into temp replacing this occurrence of target by
23                 ! substring
24                 temp(1:ipos-1) = work(1:ipos-1)
25                 DO i=1,lsub
26                     temp(i+ipos-1) = substring(i:i)
27                 ENDDO
28                 temp(ipos+lsub:) = work(ipos+lt:)
29                 work => temp ! make new version of work point at the temp space
30                 IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
31                 ! move search and replacement positions over the effected positions
32                 ipos = ipos-lt+1
33             ENDIF
34             ipos=ipos-1
35         ENDDO
36     ELSE ! forward search
37         ipos = 1; ipow = 1
38         DO
39             IF( ipos > ls-lt+1 )EXIT ! search past end of string
40             ! test for occurrence of target in string at this position
41             IF( string(ipos:ipos+lt-1) == target )THEN
42                 ! match found allocate space for string with this occurrence of
43                 ! target replaced by substring
44                 ALLOCATE(temp(1:SIZE(work)+lsub-lt))
45                 ! copy work into temp replacing this occurrence of target by
46                 ! substring
47                 temp(1:ipow-1) = work(1:ipow-1)
48                 DO i=1,lsub
49                     temp(i+ipow-1) = substring(i:i)
50                 ENDDO
51                 temp(ipow+lsub:) = work(ipow+lt:)
52                 work => temp ! make new version of work point at the temp space
53                 IF(.NOT.rep_search)EXIT ! exit if only first replacement wanted
54                 ! move search and replacement positions over the effected positions
55                 ipos = ipos+lt-1; ipow = ipow+lsub-1
56             ENDIF
57             ipos=ipos+1; ipow=ipow+1
58         ENDDO
59     ENDIF
60     replace_ccc%chars => work
61 ENDFUNCTION replace_ccc

62 !----- Remove procedures -----!
63 FUNCTION remove_s(string,start,finish)
64     type(VARYING_STRING)      :: remove_s
65     type(VARYING_STRING),INTENT(IN) :: string
66     INTEGER,INTENT(IN),OPTIONAL  :: start
67     INTEGER,INTENT(IN),OPTIONAL  :: finish
68     ! returns as result the string produced by the actions
69     ! removes the characters between start and finish from string reducing it in
70     ! size by MAX(0,ABS(finish-start+1))
71     ! if start < 1 or is missing then assumes start=1

```

```

1      ! if finish > LEN(string) or is missing then assumes finish=LEN(string)
2      CHARACTER,DIMENSION(:),POINTER :: arg_str
3      INTEGER                          :: is,if,ls
4      ls = LEN(string)
5      IF (PRESENT(start)) THEN
6          is = MAX(1,start)
7      ELSE
8          is = 1
9      ENDIF
10     IF (PRESENT(finish)) THEN
11         if = MIN(ls,finish)
12     ELSE
13         if = ls
14     ENDIF
15     IF( if < is ) THEN ! zero characters to be removed, string is unchanged
16         ALLOCATE(arg_str(1:ls))
17         arg_str = string%chars
18     ELSE
19         ALLOCATE(arg_str(1:ls-if+is-1) )
20         arg_str(1:is-1) = string%chars(1:is-1)
21         arg_str(is:) = string%chars(if+1:)
22     ENDIF
23     remove_s%chars => arg_str
24 ENDFUNCTION remove_s
25
26 FUNCTION remove_c(string,start,finish)
27     type(VARYING_STRING)      :: remove_c
28     CHARACTER(LEN=*),INTENT(IN) :: string
29     INTEGER,INTENT(IN),OPTIONAL :: start
30     INTEGER,INTENT(IN),OPTIONAL :: finish
31     ! returns as result the string produced by the actions
32     ! removes the characters between start and finish from string reducing it in
33     ! size by MAX(0,ABS(finish-start+1))
34     ! if start < 1 or is missing then assumes start=1
35     ! if finish > LEN(string) or is missing then assumes finish=LEN(string)
36     CHARACTER,DIMENSION(:),POINTER :: arg_str
37     INTEGER                          :: is,if,ls
38     ls = LEN(string)
39     IF (PRESENT(start)) THEN
40         is = MAX(1,start)
41     ELSE
42         is = 1
43     ENDIF
44     IF (PRESENT(finish)) THEN
45         if = MIN(ls,finish)
46     ELSE
47         if = ls
48     ENDIF
49     IF( if < is ) THEN ! zero characters to be removed, string is unchanged
50         ALLOCATE(arg_str(1:ls))
51         DO i=1,ls
52             arg_str(i) = string(i:i)
53         ENDDO
54     ELSE
55         ALLOCATE(arg_str(1:ls-if+is-1) )
56         DO i=1,is-1
57             arg_str(i) = string(i:i)
58         ENDDO
59         DO i=is,ls-if+is-1
60             arg_str(i) = string(i-is+if+1:i-is+if+1)
61         ENDDO
62     ENDIF
63     remove_c%chars => arg_str
64 ENDFUNCTION remove_c
65
66 !----- Extract procedures -----!
67 FUNCTION extract_s(string,start,finish)
68     type(VARYING_STRING),INTENT(IN) :: string
69     INTEGER,INTENT(IN),OPTIONAL     :: start
70     INTEGER,INTENT(IN),OPTIONAL     :: finish
71     type(VARYING_STRING)            :: extract_s
72     ! extracts the characters between start and finish from string and

```

```

1      ! delivers these as the result of the function, string is unchanged
2      ! if start < 1 or is missing then it is treated as 1
3      ! if finish > LEN(string) or is missing then it is treated as LEN(string)
4      INTEGER                                :: is,if
5      IF (PRESENT(start)) THEN
6          is = MAX(1,start)
7      ELSE
8          is = 1
9      ENDIF
10     IF (PRESENT(finish)) THEN
11         if = MIN(LEN(string),finish)
12     ELSE
13         if = LEN(string)
14     ENDIF
15     ALLOCATE(extract_s%chars(1:if-is+1))
16     extract_s%chars = string%chars(is:if)
17 ENDFUNCTION extract_s
18
19 FUNCTION extract_c(string,start,finish)
20 CHARACTER(LEN=*),INTENT(IN) :: string
21 INTEGER,INTENT(IN),OPTIONAL :: start
22 INTEGER,INTENT(IN),OPTIONAL :: finish
23 type(VARYING_STRING)        :: extract_c
24 ! extracts the characters between start and finish from character string and
25 ! delivers these as the result of the function, string is unchanged
26 ! if start < 1 or is missing then it is treated as 1
27 ! if finish > LEN(string) or is missing then it is treated as LEN(string)
28 INTEGER                                :: is,if
29 IF (PRESENT(start)) THEN
30     is = MAX(1,start)
31 ELSE
32     is = 1
33 ENDIF
34 IF (PRESENT(finish)) THEN
35     if = MIN(LEN(string),finish)
36 ELSE
37     if = LEN(string)
38 ENDIF
39 ALLOCATE(extract_c%chars(1:if-is+1))
40 DO i=is,if
41     extract_c%chars(i-is+1) = string(i:i)
42 ENDDO
43 ENDFUNCTION extract_c
44
45 !----- Split procedures -----!
46 SUBROUTINE split_s(string,word,set,separator,back)
47 type(VARYING_STRING),INTENT(INOUT)    :: string
48 type(VARYING_STRING),INTENT(OUT)      :: word
49 type(VARYING_STRING),INTENT(IN)       :: set
50 type(VARYING_STRING),INTENT(OUT),OPTIONAL :: separator
51 LOGICAL,INTENT(IN),OPTIONAL          :: back
52 ! splits the input string at the first(last) character in set
53 ! returns the leading(trailing) substring in word and the trailing(leading)
54 ! substring in string. The search is done in the forward or backward
55 ! direction depending on back. If separator is present, the actual separator
56 ! character found is returned in separator.
57 ! If no character in set is found string and separator are returned as
58 ! zero length and the whole input string is returned in word.
59 LOGICAL                                :: dir_switch
60 INTEGER                                :: ls,tpos
61 ls = LEN(string)
62 IF( PRESENT(back) )THEN
63     dir_switch = back
64 ELSE
65     dir_switch = .FALSE.
66 ENDIF
67 IF(dir_switch)THEN ! backwards search
68     DO tpos = ls,1,-1
69         IF(ANY(string%chars(tpos) == set%chars))EXIT
70     ENDDO
71     word%chars => string%chars(tpos+1:ls)
72     IF(PRESENT(separator))THEN

```

```

1         IF(tpos==0)THEN
2             separator = ""
3         ELSE
4             separator%chars => string%chars(tpos:tpos)
5         ENDIF
6     ENDIF
7     string%chars => string%chars(1:tpos-1)
8 ELSE ! forwards search
9     DO tpos =1,ls
10        IF(ANY(string%chars(tpos) == set%chars))EXIT
11    ENDDO
12    word%chars => string%chars(1:tpos-1)
13    IF(PRESENT(separator))THEN
14        IF(tpos==ls+1)THEN
15            separator = ""
16        ELSE
17            separator%chars => string%chars(tpos:tpos)
18        ENDIF
19    ENDIF
20    string%chars => string%chars(tpos+1:ls)
21 ENDIF
22 ENDSUBROUTINE split_s

23 SUBROUTINE split_c(string,word,set,separator,back)
24     type(VARYING_STRING),INTENT(INOUT)      :: string
25     type(VARYING_STRING),INTENT(OUT)        :: word
26     CHARACTER(LEN=*),INTENT(IN)            :: set
27     type(VARYING_STRING),INTENT(OUT),OPTIONAL :: separator
28     LOGICAL,INTENT(IN),OPTIONAL            :: back
29     ! splits the input string at the first(last) character in set
30     ! returns the leading(trailing) substring in word and the trailing(leading)
31     ! substring in string. The search is done in the forward or backward
32     ! direction depending on back. If separator is present, the actual separator
33     ! character found is returned in separator.
34     ! If no character in set is found string and separator are returned as
35     ! zero length and the whole input string is returned in word.
36     LOGICAL                :: dir_switch
37     INTEGER                 :: ls,tpos,lset
38     ls = LEN(string); lset = LEN(set)
39     IF( PRESENT(back) )THEN
40         dir_switch = back
41     ELSE
42         dir_switch = .FALSE.
43     ENDIF
44     IF(dir_switch)THEN ! backwards search
45         BSEARCH:DO tpos = ls,1,-1
46             DO i=1,lset
47                 IF(string%chars(tpos) == set(i:i))EXIT BSEARCH
48             ENDDO
49         ENDDO BSEARCH
50         word%chars => string%chars(tpos+1:ls)
51         IF(PRESENT(separator))THEN
52             IF(tpos==0)THEN
53                 separator = ""
54             ELSE
55                 separator%chars => string%chars(tpos:tpos)
56             ENDIF
57         ENDIF
58         string%chars => string%chars(1:tpos-1)
59     ELSE ! forwards search
60         FSEARCH:DO tpos =1,ls
61             DO i=1,lset
62                 IF(string%chars(tpos) == set(i:i))EXIT FSEARCH
63             ENDDO
64         ENDDO FSEARCH
65         word%chars => string%chars(1:tpos-1)
66         IF(PRESENT(separator))THEN
67             IF(tpos==ls+1)THEN
68                 separator = ""
69             ELSE
70                 separator%chars => string%chars(tpos:tpos)
71             ENDIF

```

```

1      ENDIF
2      string%chars => string%chars(tpos+1:ls)
3      ENDIF
4      ENDSUBROUTINE split_c

5      !----- INDEX procedures -----!
6      FUNCTION index_ss(string,substring,back)
7          type(VARYING_STRING),INTENT(IN) :: string,substring
8          LOGICAL,INTENT(IN),OPTIONAL    :: back
9          INTEGER                        :: index_ss
10         ! returns the starting position in string of the substring
11         ! scanning from the front or back depending on the logical argument back
12         LOGICAL                        :: dir_switch
13         INTEGER                        :: ls,lsub
14         ls = LEN(string); lsub = LEN(substring)
15         IF( PRESENT(back) )THEN
16             dir_switch = back
17         ELSE
18             dir_switch = .FALSE.
19         ENDIF
20         IF(dir_switch)THEN ! backwards search
21             DO i = ls-lsub+1,1,-1
22                 IF( ALL(string%chars(i:i+lsub-1) == substring%chars) )THEN
23                     index_ss = i
24                     RETURN
25                 ENDIF
26             ENDDO
27             index_ss = 0
28         ELSE ! forward search
29             DO i = 1,ls-lsub+1
30                 IF( ALL(string%chars(i:i+lsub-1) == substring%chars) )THEN
31                     index_ss = i
32                     RETURN
33                 ENDIF
34             ENDDO
35             index_ss = 0
36         ENDIF
37     ENDFUNCTION index_ss

38
39     FUNCTION index_sc(string,substring,back)
40         type(VARYING_STRING),INTENT(IN) :: string
41         CHARACTER(LEN=*),INTENT(IN)    :: substring
42         LOGICAL,INTENT(IN),OPTIONAL    :: back
43         INTEGER                        :: index_sc
44         ! returns the starting position in string of the substring
45         ! scanning from the front or back depending on the logical argument back
46         LOGICAL                        :: dir_switch,matched
47         INTEGER                        :: ls,lsub
48         ls = LEN(string); lsub = LEN(substring)
49         IF( PRESENT(back) )THEN
50             dir_switch = back
51         ELSE
52             dir_switch = .FALSE.
53         ENDIF
54         IF( dir_switch ) THEN ! backwards search
55             DO i = ls-lsub+1,1,-1
56                 matched = .TRUE.
57                 DO j = 1,ls
58                     IF( string%chars(i+j-1) /= substring(j:j) )THEN
59                         matched = .FALSE.
60                         EXIT
61                     ENDIF
62                 ENDDO
63                 IF( matched )THEN
64                     index_sc = i
65                     RETURN
66                 ENDIF
67             ENDDO
68             index_sc = 0
69         ELSE ! forward search
70             DO i = 1,ls-lsub+1
71                 matched = .TRUE.

```



```

1      DO j = 1, lsub
2          IF( string%chars(i+j-1) /= substring(j:j) )THEN
3              matched = .FALSE.
4              EXIT
5          ENDDIF
6      ENDDO
7      IF( matched )THEN
8          index_sc = i
9          RETURN
10         ENDDIF
11     ENDDO
12     index_sc = 0
13 ENDFUNCTION index_sc

14
15
16 FUNCTION index_cs(string, substring, back)
17     CHARACTER(LEN=*) , INTENT(IN)      :: string
18     type(VARYING_STRING), INTENT(IN)  :: substring
19     LOGICAL, INTENT(IN), OPTIONAL     :: back
20     INTEGER                          :: index_cs
21     ! returns the starting position in string of the substring
22     ! scanning from the front or back depending on the logical argument back
23     LOGICAL                          :: dir_switch, matched
24     INTEGER                          :: ls, lsub
25     ls = LEN(string); lsub = LEN(substring)
26     IF( PRESENT(back) )THEN
27         dir_switch = back
28     ELSE
29         dir_switch = .FALSE.
30     ENDDIF
31     IF(dir_switch)THEN ! backwards search
32         DO i = ls-lsub+1, 1, -1
33             matched = .TRUE.
34             DO j = 1, lsub
35                 IF( string(i+j-1:i+j-1) /= substring%chars(j) )THEN
36                     matched = .FALSE.
37                     EXIT
38                 ENDDIF
39             ENDDO
40             IF( matched )THEN
41                 index_cs = i
42                 RETURN
43             ENDDIF
44         ENDDO
45         index_cs = 0
46     ELSE ! forward search
47         DO i = 1, ls-lsub+1
48             matched = .TRUE.
49             DO j = 1, lsub
50                 IF( string(i+j-1:i+j-1) /= substring%chars(j) )THEN
51                     matched = .FALSE.
52                     EXIT
53                 ENDDIF
54             ENDDO
55             IF( matched )THEN
56                 index_cs = i
57                 RETURN
58             ENDDIF
59         ENDDO
60         index_cs = 0
61     ENDDIF
62 ENDFUNCTION index_cs

63
64 !----- SCAN procedures -----!
65 FUNCTION scan_ss(string, set, back)
66     type(VARYING_STRING), INTENT(IN)  :: string, set
67     LOGICAL, INTENT(IN), OPTIONAL     :: back
68     INTEGER                          :: scan_ss
69     ! returns the first position in string occupied by a character from
70     ! the characters in set, scanning is forward or backwards depending on back
71     LOGICAL                          :: dir_switch
72     INTEGER                          :: ls

```

```

1      CHARACTER::tmp ! inserted to work round a temporary bug in F90 1.1
2      ls = LEN(string)
3      IF( PRESENT(back) )THEN
4          dir_switch = back
5      ELSE
6          dir_switch = .FALSE.
7      ENDIF
8      IF(dir_switch)THEN ! backwards search
9          DO i = ls,1,-1
10         tmp=string%chars(i) ! bug work round
11             IF( ANY( set%chars == tmp ) )THEN
12                 scan_ss = i
13                 RETURN
14             ENDIF
15         ENDDO
16         scan_ss = 0
17     ELSE ! forward search
18         DO i = 1,ls
19         tmp=string%chars(i) ! bug work round
20             IF( ANY( set%chars == tmp ) )THEN
21                 scan_ss = i
22                 RETURN
23             ENDIF
24         ENDDO
25         scan_ss = 0
26     ENDIF
27 ENDFUNCTION scan_ss
28
29 FUNCTION scan_sc(string,set,back)
30     type(VARYING_STRING),INTENT(IN) :: string
31     CHARACTER(LEN=*),INTENT(IN)    :: set
32     LOGICAL,INTENT(IN),OPTIONAL     :: back
33     INTEGER                          :: scan_sc
34     ! returns the first position in string occupied by a character from
35     ! the characters in set, scanning is forward or backwards depending on back
36     LOGICAL                          :: dir_switch,matched
37     INTEGER                          :: ls
38     ls = LEN(string)
39     IF( PRESENT(back) )THEN
40         dir_switch = back
41     ELSE
42         dir_switch = .FALSE.
43     ENDIF
44     IF(dir_switch)THEN ! backwards search
45         DO i = ls,1,-1
46             matched = .FALSE.
47             DO j = 1,LEN(set)
48                 IF( string%chars(i) == set(j:j) )THEN
49                     matched = .TRUE.
50                     EXIT
51                 ENDIF
52             ENDDO
53             IF( matched )THEN
54                 scan_sc = i
55                 RETURN
56             ENDIF
57         ENDDO
58         scan_sc = 0
59     ELSE ! forward search
60         DO i = 1,ls
61             matched = .FALSE.
62             DO j = 1,LEN(set)
63                 IF( string%chars(i) == set(j:j) )THEN
64                     matched = .TRUE.
65                     EXIT
66                 ENDIF
67             ENDDO
68             IF( matched )THEN
69                 scan_sc = i
70                 RETURN
71             ENDIF
72         ENDDO

```

```

1      scan_sc = 0
2      ENDIF
3      ENDFUNCTION scan_sc
4
5      FUNCTION scan_cs(string,set,back)
6      CHARACTER(LEN=*),INTENT(IN)      :: string
7      type(VARYING_STRING),INTENT(IN)  :: set
8      LOGICAL,INTENT(IN),OPTIONAL      :: back
9      INTEGER                          :: scan_cs
10     ! returns the first position in character string occupied by a character from
11     ! the characters in set, scanning is forward or backwards depending on back
12     LOGICAL                          :: dir_switch,matched
13     INTEGER                          :: ls
14     ls = LEN(string)
15     IF( PRESENT(back) )THEN
16         dir_switch = back
17     ELSE
18         dir_switch = .FALSE.
19     ENDIF
20     IF(dir_switch)THEN ! backwards search
21         DO i = ls,1,-1
22             matched = .FALSE.
23             DO j = 1,LEN(set)
24                 IF( string(i:i) == set%chars(j) )THEN
25                     matched = .TRUE.
26                     EXIT
27                 ENDIF
28             ENDDO
29             IF( matched )THEN
30                 scan_cs = i
31                 RETURN
32             ENDIF
33         ENDDO
34         scan_cs = 0
35     ELSE ! forward search
36         DO i = 1,ls
37             matched = .FALSE.
38             DO j = 1,LEN(set)
39                 IF( string(i:i) == set%chars(j) )THEN
40                     matched = .TRUE.
41                     EXIT
42                 ENDIF
43             ENDDO
44             IF( matched )THEN
45                 scan_cs = i
46                 RETURN
47             ENDIF
48         ENDDO
49         scan_cs = 0
50     ENDIF
51     ENDFUNCTION scan_cs
52
53     !----- VERIFY procedures -----!
54     FUNCTION verify_ss(string,set,back)
55     type(VARYING_STRING),INTENT(IN)  :: string,set
56     LOGICAL,INTENT(IN),OPTIONAL      :: back
57     INTEGER                          :: verify_ss
58     ! returns the first position in string not occupied by a character from
59     ! the characters in set, scanning is forward or backwards depending on back
60     LOGICAL                          :: dir_switch
61     INTEGER                          :: ls
62     CHARACTER::tmp ! F90 1.1 bug work round
63     ls = LEN(string)
64     IF( PRESENT(back) )THEN
65         dir_switch = back
66     ELSE
67         dir_switch = .FALSE.
68     ENDIF
69     IF(dir_switch)THEN ! backwards search
70         DO i = ls,1,-1
71             tmp=string%chars(i) ! bug work round
72             IF( .NOT.(ANY( set%chars == tmp )) )THEN

```

```

1         verify_ss = i
2         RETURN
3     ENDIF
4 ENDDO
5     verify_ss = 0
6 ELSE ! forward search
7     DO i = 1,ls
8 tmp=string%chars(i) ! bug work round
9         IF( .NOT.(ANY( set%chars == tmp )) )THEN
10            verify_ss = i
11            RETURN
12        ENDIF
13    ENDDO
14    verify_ss = 0
15 ENDIF
16 ENDFUNCTION verify_ss
17
18 FUNCTION verify_sc(string,set,back)
19 type(VARYING_STRING),INTENT(IN) :: string
20 CHARACTER(LEN=*),INTENT(IN)    :: set
21 LOGICAL,INTENT(IN),OPTIONAL    :: back
22 INTEGER                        :: verify_sc
23 ! returns the first position in string not occupied by a character from
24 ! the characters in set, scanning is forward or backwards depending on back
25 LOGICAL                        :: dir_switch
26 INTEGER                        :: ls
27 ls = LEN(string)
28 IF( PRESENT(back) )THEN
29     dir_switch = back
30 ELSE
31     dir_switch = .FALSE.
32 ENDIF
33 IF(dir_switch)THEN ! backwards search
34     back_string_search:DO i = ls,1,-1
35         DO j = 1,LEN(set)
36             IF( string%chars(i) == set(j:j) )CYCLE back_string_search
37             ! cycle string search if string character found in set
38         ENDDO
39         ! string character not found in set index i is result
40         verify_sc = i
41         RETURN
42     ENDDO back_string_search
43     ! each string character found in set
44     verify_sc = 0
45 ELSE ! forward search
46     frwd_string_search:DO i = 1,ls
47         DO j = 1,LEN(set)
48             IF( string%chars(i) == set(j:j) )CYCLE frwd_string_search
49         ENDDO
50         verify_sc = i
51         RETURN
52     ENDDO frwd_string_search
53     verify_sc = 0
54 ENDIF
55 ENDFUNCTION verify_sc
56
57 FUNCTION verify_cs(string,set,back)
58 CHARACTER(LEN=*),INTENT(IN)    :: string
59 type(VARYING_STRING),INTENT(IN) :: set
60 LOGICAL,INTENT(IN),OPTIONAL    :: back
61 INTEGER                        :: verify_cs
62 ! returns the first position in icharacter string not occupied by a character
63 ! from the characters in set, scanning is forward or backwards depending on
64 ! back
65 LOGICAL                        :: dir_switch
66 INTEGER                        :: ls
67 ls = LEN(string)
68 IF( PRESENT(back) )THEN
69     dir_switch = back
70 ELSE
71     dir_switch = .FALSE.
72 ENDIF

```

```

1      IF(dir_switch)THEN ! backwards search
2      back_string_search:DO i = ls,1,-1
3          DO j = 1,LEN(set)
4              IF( string(i:i) == set%chars(j) )CYCLE back_string_search
5          ENDDO
6              verify_cs = i
7              RETURN
8      ENDDO back_string_search
9      verify_cs = 0
10     ELSE ! forward search
11     frwd_string_search:DO i = 1,ls
12         DO j = 1,LEN(set)
13             IF( string(i:i) == set%chars(j) )CYCLE frwd_string_search
14         ENDDO
15             verify_cs = i
16             RETURN
17     ENDDO frwd_string_search
18     verify_cs = 0
19     ENDIF
20     ENDFUNCTION verify_cs
21
22     !----- LEN_TRIM procedure -----!
23     FUNCTION len_trim_s(string)
24     type(VARYING_STRING),INTENT(IN) :: string
25     INTEGER                          :: len_trim_s
26     ! Returns the length of the string without counting trailing blanks
27     INTEGER                          :: ls
28     ls=LEN(string)
29     len_trim_s = 0
30     DO i = ls,1,-1
31         IF (string%chars(i) /= BLANK) THEN
32             len_trim_s = i
33             EXIT
34         ENDIF
35     ENDDO
36     ENDFUNCTION len_trim_s
37
38     !----- TRIM procedure -----!
39     FUNCTION trim_s(string)
40     type(VARYING_STRING),INTENT(IN) :: string
41     type(VARYING_STRING)           :: trim_s
42     ! Returns the argument string with trailing blanks removed
43     INTEGER                        :: ls,pos
44     ls=LEN(string)
45     pos=0
46     DO i = ls,1,-1
47         IF(string%chars(i) /= BLANK) THEN
48             pos=i
49             EXIT
50         ENDIF
51     ENDDO
52     ALLOCATE(trim_s%chars(1:pos))
53     trim_s%chars(1:pos) = string%chars(1:pos)
54     ENDFUNCTION trim_s
55
56     !----- IACHAR interface -----!
57     FUNCTION iachar_s(string)
58     type(VARYING_STRING),INTENT(IN) :: string
59     INTEGER                          :: iachar_s
60     ! returns the position of the character string in the ISO 646
61     ! collating sequence.
62     ! string must be of length one
63     IF (LEN(string) /= 1) THEN
64         WRITE(*,*) " ERROR, argument in IACHAR not of length one"
65     STOP
66     ENDIF
67     iachar_s = IACHAR(string%chars(1))
68     ENDFUNCTION iachar_s
69
70     !----- ICHAR procedure -----!
71     FUNCTION ichar_s(string)
72     type(VARYING_STRING),INTENT(IN) :: string

```

```

1      INTEGER                :: ichar_s
2      ! returns the position of character from string in the processor collating
3      ! sequence.
4      ! string must be of length one
5      IF (LEN(string) /= 1) THEN
6          WRITE(*,*) " Argument string in ICHAR has to be of length one"
7          STOP
8      ENDIF
9      ichar_s = ICHAR(string%chars(1))
10     ENDFUNCTION ichar_s
11
12     !----- ADJUSTL procedure -----!
13     FUNCTION adjustl_s(string)
14         type(VARYING_STRING),INTENT(IN) :: string
15         type(VARYING_STRING)           :: adjustl_s
16         ! Returns the string adjusted to the left, removing leading blanks and
17         ! inserting trailing blanks
18         INTEGER                        :: ls,pos
19         ls=LEN(string)
20         DO pos = 1,ls
21             IF(string%chars(pos) /= blank) EXIT
22         ENDDO
23         ! pos now holds the position of the first non-blank character
24         ! or ls+1 if all characters are blank
25         ALLOCATE(adjustl_s%chars(1:ls))
26         adjustl_s%chars(1:ls-pos+1) = string%chars(pos:ls)
27         adjustl_s%chars(ls-pos+2:ls) = blank
28     ENDFUNCTION adjustl_s
29
30     !----- ADJUSTR procedure -----!
31     FUNCTION adjustr_s(string)
32         type(VARYING_STRING),INTENT(IN) :: string
33         type(VARYING_STRING)           :: adjustr_s
34         ! Returns the string adjusted to the right, removing trailing blanks
35         ! and inserting leading blanks
36         INTEGER                        :: ls,pos
37         ls=LEN(string)
38         DO pos = ls,1,-1
39             IF(string%chars(pos) /= blank) EXIT
40         ENDDO
41         ! pos now holds the position of the last non-blank character
42         ! or zero if all characters are blank
43         ALLOCATE(adjustr_s%chars(1:ls))
44         adjustr_s%chars(ls-pos+1:ls) = string%chars(1:pos)
45         adjustr_s%chars(1:ls-pos) = blank
46     ENDFUNCTION adjustr_s
47
48     ENDMODULE ISO_VARYING_STRING

```

Annex B

(Informative)

This annex includes some examples illustrating the use of facilities conformant with this International Standard. It should be noted that while every care has been taken by the technical working group to ensure that these example programs are a correct implementation of the stated problems using this International Standard and in valid Fortran code, no guarantee is given or implied that this code will produce correct results, or even that it will execute on any particular processor.

```

8 PROGRAM word_count
9 !-----!
10 ! Counts the number of "words" contained in a file. The words are assumed to !
11 ! be terminated by any one of: !
12 ! space,comma,period,!,?, or the EoR !
13 ! The file may have records of any length and the file may contain any number !
14 ! of records. !
15 ! The program prompts for the name of the file to be subject to a word count !
16 ! and the result is written to the default output unit !
17 !-----!
18 USE ISO_VARYING_STRING
19 type(VARYING_STRING) :: line, fname
20 INTEGER :: ierr, nd, wcount=0
21 WRITE(*,ADVANCE='NO',FMT='(A)') " Input name of file?"
22 CALL GET(String=fname) ! read the required filename from the default
23 ! input unit assumed to be the whole of the record read
24 OPEN(UNIT=1,FILE=CHAR(fname)) ! CHAR(fname) converts to the type
25 ! required by FILE= specifier
26 file_read: DO ! until EOF reached
27 CALL GET(1,line,IOSTAT=ierr) ! read next line of file
28 IF(ierr == -1)EXIT file_read
29 word_scan: DO ! until end of line
30 nd=SCAN(line," ,.!?" ) ! scan to find end of word
31 IF(nd == 0)THEN ! EoR is end of word
32 nd = LEN(line)
33 EXIT word_scan
34 ENDIF
35 IF(nd > 1) wcount=wcount+1 ! at least one non-terminator character
36 ! in the word
37 line = REMOVE(line,1,nd) ! strips the counted word and its terminator
38 ! from the line reducing its length before
39 ! rescanning for the next word
40 ENDDO word_scan
41 IF(nd > 0) wcount=wcount+1
42 ENDDO file_read
43 IF(ierr < 0)THEN
44 WRITE(*,*) "No. of words in file =",wcount
45 ELSEIF(ierr > 0)THEN
46 WRITE(*,*) "Error in GET file in word_count, No. ",ierr
47 ENDIF
48 ENDPROGRAM word_count

```

Note, it is not claimed that the above program is the best way to code this problem, nor even that it is a good way, merely that it is a way of solving this simple problem using the facilities defined by this International Standard.

A second and rather more realistic example is one which extends the above trivial example by producing a full vocabulary list along with frequency of occurrence for each different word.

```

54 PROGRAM vocabulary_word_count
55 !-----!
56 ! Counts the number of "words" contained in a file. The words are assumed to !
57 ! be terminated by any one of: !

```

```

1      ! space,comma,period,!,?, or the EoR                                     !
2      ! The file may have records of any length and the file may contain any number !
3      ! of records.                                                            !
4      ! The program prompts for the name of the file to be subject to a word count !
5      ! and the result is written to the default output unit                   !
6      ! Also builds a list of the vocabulary found and the frequency of occurrence !
7      ! of each different word.                                               !
8      !-----!
9      USE ISO_VARYING_STRING
10     type(VARYING_STRING) :: line,word,fname
11     INTEGER               :: ierr,nd,wcount=0
12     !-----!
13     ! Vocabulary list and frequency count arrays. The size of these arrays will !
14     ! be extended dynamically in steps of 100 as the used vocabulary grows    !
15     !-----!
16     type(VARYING_STRING),ALLOCATABLE,DIMENSION(:) :: vocab
17     INTEGER,ALLOCATABLE,DIMENSION(:)              :: freq
18     INTEGER                                       :: list_size=100,list_top=0
19     !-----!
20     ! Initialise the lists and determine the file to be processed              !
21     !-----!
22     ALLOCATE(vocab(1:list_size),freq(1:list_size))
23     WRITE(*,ADVANCE='NO',FMT='(A)') " Input name of file?"
24     CALL GET(String=fname) ! read the required filename from the default
25     ! input unit assumed to be the whole of the record read
26     OPEN(UNIT=1,FILE=CHAR(fname)) ! CHAR(fname) converts to the type
27     ! required by FILE= specifier
28     file_read: DO ! until EOF reached
29         CALL GET(1,line,IOSTAT=ierr) ! read next line of file
30         IF(ierr == -1)EXIT file_read
31         word_scan: DO ! until end of line
32             nd=SCAN(line," ,.!?" ) ! scan to find end of word
33             IF(nd == 0)THEN ! EoR is end of word
34                 nd = LEN(line)
35                 EXIT word_scan
36             ENDIF
37             IF(nd > 1)THEN ! at least one non-terminator character in the word
38                 wcount=wcount+1
39                 word = EXTRACT(line,1,nd-1)
40                 CALL update_vocab_lists
41             ENDIF
42             line = REMOVE(line,1,nd) ! strips the counted word and its terminator
43             ! from the line reducing its length before
44             ! rescanning for the next word
45         ENDDO word_scan
46         IF(nd > 0)THEN ! at least one character in the word
47             wcount=wcount+1
48             word = EXTRACT(line,1,nd-1)
49             CALL update_vocab_lists
50         ENDIF
51     ENDDO file_read
52     IF(ierr < 0)THEN
53         WRITE(*,*) "No. of words in file =",wcount
54         WRITE(*,*) "There are ",list_top," distinct words"
55         WRITE(*,*) "with the following frequencies of occurrence"
56         print_loop: DO i=1,list_top
57             WRITE(*,FMT='(1X,I6,2X)',ADVANCE='NO') freq(i)
58             CALL PUT_LINE(String=vocab(i))
59         ENDDO print_loop
60     ELSEIF(ierr > 0)THEN
61         WRITE(*,*) "Error in GET in vocabulary_word_count, No.",ierr
62     ENDIF
63
64     CONTAINS
65
66     SUBROUTINE extend_lists
67     !-----!
68     ! Accesses the host variables:                                           !
69     ! type(VARYING_STRING),ALLOCATABLE,DIMENSION(:) :: vocab                !
70     ! INTEGER,ALLOCATABLE,DIMENSION(:)              :: freq                  !
71     ! INTEGER                                       :: list_size            !
72     ! so as to extend the size of the lists preserving the existing vocabulary !

```



```

1      ! and frequency information in the new extended lists                                     !
2      !-----!
3      type(VARYING_STRING),DIMENSION(list_size) :: vocab_swap
4      INTEGER,DIMENSION(list_size)           :: freq_swap
5      INTEGER,PARAMETER :: list_increment=100
6      INTEGER           :: new_list_size,alerr
7      vocab_swap = vocab  ! copy old list into temporary space
8      freq_swap =freq
9      new_list_size = list_size + list_increment
10     DEALLOCATE(vocab,freq)
11     ALLOCATE(vocab(1:new_list_size),freq(1:new_list_size),STAT=alerr)
12     IF(alerr /= 0)THEN
13         WRITE(*,*) "Unable to extend vocabulary list"
14         STOP
15     ENDIF
16     vocab(1:list_size) = vocab_swap  ! copy old list back into bottom
17     freq(1:list_size) = freq_swap  ! of new extended list
18     list_size = new_list_size
19     ENDSUBROUTINE extend_lists

20     SUBROUTINE update_vocab_lists
21     !-----!
22     ! Accesses the host variables:                                     !
23     ! type(VARYING_STRING),ALLOCATABLE,DIMENSION(:) :: vocab         !
24     ! INTEGER,ALLOCATABLE,DIMENSION(:)           :: freq           !
25     ! INTEGER                                     :: list_size,list_top !
26     ! type(VARYING_STRING)                       :: word           !
27     ! searches the existing words in vocab to find a match for word  !
28     ! if found increments the freq if not found adds word to       !
29     ! list_top + 1 vocab list and sets corresponding freq to 1     !
30     ! if list_size exceeded extend the list size before updating  !
31     !-----!
32     list_search: DO i=1,list_top
33         IF(word == vocab(i))THEN
34             freq(i) = freq(i) + 1
35         RETURN
36     ENDIF
37     ENDDO list_search
38     IF(list_top == list_size)THEN
39         CALL extend_lists
40     ENDIF
41     list_top = list_top + 1
42     vocab(list_top) = word
43     freq(list_top) = 1
44     ENDSUBROUTINE update_vocab_lists

45     ENDPROGRAM vocabulary_word_count

```