Suggestion:

Improved String-handling in Fortran

Clive Page
2015 October 1

Why Bother?

The Fortran programs that I write typically spend ~99% of the time crunching numbers, but nearly all of them include at least a little string-handling, such as for

- Parsing command-line arguments
- Handling file-names and paths, e.g. creating name of a new output file from that of input
- Handling data files if they are plain-text or CSV
- Processing metadata in files, such as data identifiers, operating modes, dates & times, etc.
- Handling commands or parameters from the user's keyboard
- Dealing with error messages arising from IOMSG etc.
- Formatting data to be viewed by the user

These string-handling bits are a small part of the total but often rather messy, because there usually several ways of doing what you want, none of them entirely adequate.

Current Options

At present the programmer has three options for handling character strings:

- **CHARACTER type** – introduced in Fortran 77 (with new intrinsics added in Fortran 90/95)
- **ISO_VARYING_STRING** – introduced with Fortran 90 in document ISO/IEC 1539-2:1994 (with minor revisions in 2000)
- **CHARACTER with allocatable length** – introduced in Fortran 2003.

These are at best only semi-compatible with each other.

In my opinion none of them provides all the facilities that are really desirable.

CHARACTER type

Every character variable has to have its length declared i.e. fixed at compile-time.

This inflexibility is moderated by features such as: automatic padding with spaces, TRIM intrinsic, and passed-length dummy arguments, so that it more-or-less behaves as a dynamic string type subject to a length limit.

Main Limitations:

- The programmer has to choose an adequate length for each variable, e.g. think of the longest you can possible need and then double it, and then hope nothing goes wrong.
  - This is rather like sizing arrays before allocatable/automatic/pointer arrays became available. File-names including paths can be over 100 characters long, is 1000 sufficient? Your guess is as good as mine.
- Assignment to a variable of shorter length truncates silently – if it just removes spaces it might be what you want, but there is no warning when losing significant text, so your results might just be wrong.
  - No compiler that I know of provides a run-time diagnostic for such cases (compare the case of arrays where most compilers have an option for array-bound checking).
  - Well-engineered code ***ought*** to include checks for truncation of significant parts of strings, but it is tedious to do this, and it rarely if ever seems to be done in practice.
- The TRIM() intrinsic has to be used very frequently to avoid including excessive trailing spaces, e.g.

```
output_file = trim(input_path) // trim(input_file) // ".new"
```

CHARACTER type: minor problems

- If TRIM or LEN_TRIM are used it is hard to handle strings with significant trailing spaces.

- There are two ways of declaring string length:

```
CHARACTER :: input_file*1000, output_file*1000      ! old way
CHARACTER(LEN=1000) :: input_file, output_file      ! new way
```

- In an array of strings every element has the same length. So this is illegal code:

```
CHARACTER(LEN=9), PARAMETER :: day(7) = ['Monday', 'Tuesday', &
 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

But DATA effectively does a separate assignment to each array element so this is valid:

```
CHARACTER(LEN=9) :: day(7)
DATA day / 'Monday', 'Tuesday', &
 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'/
```

ISO_VARYING STRING

Introduced as in addendum to the Fortran 90 Standard to demonstrate that a user-defined dynamic string type could be constructed. Memory leaks were possible in the original implementation – fixed in Fortran 95.

A module defines a VARYING_STRING type and overloads all intrinsic functions and the concatenation and assignment operators so that varying strings can be used in expressions and assignment statements very much as character type. In **arrays** of VARYING_STRING each element has a dynamic length.

Limitations:

- Cannot use them in READ/WRITE either in the data-transfer list or as an internal-file value (except for WRITE with a CHAR() wrapper). New functions like GET_LINE and PUT_LINE are provided for simple whole-line reading and writing, but these are for formatted sequential advancing I/O only.
- When passing a VARYING_STRING actual argument to a procedure with a dummy argument of CHARACTER type, one can use the overloaded CHAR function for INTENT(IN) arguments; otherwise interfacing with existing procedures is more complicated.
- The long-established **sub-string notation** of CHARACTER type cannot be used (but additional procedures are provided like EXTRACT, INSERT, REMOVE, REPLACE to do this).

Allocatable Length Character

Introduced in Fortran 2003 and now supported by many modern compilers including gfortran.

Good Features:

- Scalars can be used as expected in expressions and assignments with automatic assignment of length.

  Example of use:

```
character (len=:), allocatable :: s1
s1 = 'first  '                      ! Automatic allocation on assignment
print *, '[', s1, ']'
s1 = 'longer string'                ! Automatic re-assignment of length
```

  ! But note that with this syntax:

```
s1(:) = 'new string'                ! Unchanged length with padding/truncation
```

- Can be used as expected in I/O lists of WRITE statements (and as unit item of internal-file READ).
- Combinations of allocatable-length and fixed-length strings present no problems and one can use sub-string notation and all intrinsic functions without any change in programming practice.
- Handling of strings with significant trailing spaces is now easy.

Allocatable-length character type – some limitations:

- Care is needed in **procedure calls:**
  - If the dummy argument is passed-length character (or fixed-length) - the actual argument may be an allocatable-length string provided it is allocated with sufficient length, This applies both to output arguments and input – the length of the actual argument cannot be changed by the procedure call.
  - If the dummy argument has allocatable-length then the actual argument must also have allocatable-length (much as one might expect) and then the length can be changed within the procedure.
- There is no automatic allocation of length when the variable appears in the data transfer list of a READ statement (or as the unit specifier of an internal-file WRITE). An allocation to a suitable fixed length must be done in advance.
- In **allocatable-length arrays** all elements have the same length, and there is no automatic allocation on assignment.
  - Arrays require an explicit allocation, like this:

```
character(len=:), allocatable :: array(:)
allocate( character(len=100) :: array(20) )
```

  - And, of course, if you want to change the length you need to de-allocate and re-allocate.

Summary

- The original CHARACTER type has limited flexibility; fixed-length strings are risky because of the silent truncation on assignment and in other situations when length mismatches may occur, e.g. in procedure calls.

- The ISO_VARYING_STRING provides fully dynamic length for scalars and array-elements but they are easy to use only in expressions and assignments. Since it is a derived not an intrinsic type extra code is generally needed in I/O statements, and in calls to existing procedures with CHARATER arguments.

- The new allocatable-length string is very useful for scalar applications, but automatic length changes apply only in assignment statements, and this does not extend to arrays. This is a pity, as in Fortran most scalar facilities extend naturally to arrays, indeed it is often said that "*arrays are first-class objects*".

So: what should we teach new users of Fortran about handling strings? Should they learn about all of these facilities, or only some of them? I really don't know.

A possible way forward

I think we need a fully-dynamic character string type, such that when you have an array of them, each element can have its own length, and that the string length should be set or re-set in all situations when the string value is modified.

In an ideal world this would be retrofitted to the existing CHARACTER type, but I suspect it would be very hard to do this without producing many incompatibilities with existing code. In particular, finding a notation for the declaration of a fully-dynamic character type would be difficult, since

- CHARACTER(LEN=*) is already used for passed-length procedure arguments (and constants)
- CHARACTER(LEN=:) is already used for allocatable-length scalar strings.

My suggestion is that the facilities of ISO_VARYING_STRING should be built in to the language, such that VARYING_STRING would be an additional intrinsic type, which could be used in all I/O contexts, and allow use of the existing sub-string notation.

In the longer term an intrinsic VARYING_STRING type might entirely supersede CHARACTER, but the latter should probably be preserved for compatibility reasons.